[redacted]

Report No. 458

Math

THE AUTOMATIC GENERATION OF DETERMINISTIC PARSING ALGORITHMS

by

Alan James Beals

June 21, 1971

ILLIAC IV Document No. 248

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. 458

THE AUTOMATIC GENERATION OF DETERMINISTIC PARSING ALGORITHMS

by

Alan James Beals

May 25, 1971

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

This work was supported in part by the Advanced Research Projects Agency as administered by the Rome Air Development Center under Contract No. USAF 30(602)-4144 and submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science, 1971.

ABSTRACT

This paper describes an algorithm for the conversion of a grammar in the form of a set of context free productions into a deterministic parsing algorithm described by a set of modified Floyd productions. The algorithm is extended in such a way that it may easily become a part of a complete translator writing system and make use of the information available in the semantic part of such a system. The paper also includes a determination of the subclass of context free grammars which can be successfully converted and a comparison of this algorithm with some other methods of generating parsing algorithms from context free grammars.

TABLE OF CONTENTS

	Page
1. INTRODUCTION	1
2. NOTATION AND DEFINITIONS	5
3. THE BASIC ALGORITHM	10
3.1 Label (Group) Determination	11
3.2 Descriptor Set Generation	12
3.3 Descriptor to FPL Production Mapping	13
3.4 Preclusion and Error Detection	15
3.5 Summary and Example	15
3.5.1 Input Syntax	16
3.5.2 Labels (Groups) Needed	16
3.5.3 Descriptor Sets	16
3.5.4 Resulting FPL Production Set	17
4. COMBINATION OF FPL PRODUCTIONS	18
4.1 CT(m) Groups	19
4.2 Marker Symbol Combination	21
4.3 Initial CF Grammar Revision	25
4.4 Summary and Example	27
4.4.1 Input Syntax	28
4.4.2 Grammar Revisions	29
4.4.3 FPL Production Subset	29
4.4.4 Combinations	30
5. LOOKAHEAD CONSTRUCTION	31

Page

6. CF GRAMMARS ACCEPTED 39

6.1 Strong LR (k) Grammars 39
6.2 Explicit Lookahead Adequacy 44
6.3 Implicit Lookback Adequacy 49
6.4 Conclusion 57

7. RELATIVE EFFECTIVENESS 58

7.1 Upper Bounds. 58

7.1.1 FPL Production Set 60
7.1.2 Comparison 62

7.2 Average Times 63
7.3 Subjective Evaluation 65
7.4 Summary 68

8. THE IMPLEMENTED TWS 70

8.1 Syntax 70
8.2 Semantics 75
8.3 Other Considerations 79
8.4 Summary 81

9. CONCLUSION 83

APPENDICES 87

A. TWINKLE SYNTAX OF DEMALGL 87
B. PRØTAB OF DEMALGL. 90
C. FPL PRODUCTION SET FOR DEMALGL 97
D. IMPLEMENTATION FLOW CHARTS 139

LIST OF REFERENCES 151

VITA 153

LIST OF FIGURES

Figure | Page

1. Flow Chart for Lookahead Buffer Usage. 33
2. Flow Chart for Lookahead Generation. 35

1. INTRODUCTION

It is the opinion here that one of the major restrictions to getting maximum use out of any computer is the lack of programming ability on the part of those people who have problems in need of solution. The primary cause of this inability is the unavailability of problem oriented programming languages. This frequently leads to one or more of the following: the problem solver is uncertain about capabilities and features of the specific machine and general purpose language available and thus fails to use the computer in many cases where it could be extremely valuable, or he does use the machine, but writes his own programs in a relatively simple and inefficient manner, or finally, he turns the problem over to a programmer who has the ability to get maximum use of the machine but does not have a thorough understanding of all the aspects of the problem, and thus can computerize only those specific aspects of a problem given him.

The solution to this is the development of programming languages which correspond as closely as possible to the notation and terminology of specific problem areas. At the present time, very little is being done along these lines, partly because of the large effort required to implement a compiler or translator for a new language. The man hours required for such projects can seldom be justified when the results will achieve the relatively limited usage generally accorded a specific problem oriented language. The obvious answer is to mechanize the task of compiler writing through the use of a translator writing system (TWS) whose input is the specification

of a programming language and a machine, and whose output is a compiler for that language which generates executable code for the specified machine.

Such a TWS built compiler has its task broken into three relatively distinct parts. The first is to recognize the validity and, more importantly, the structure of an input program. This is referred to as the syntactic or parsing phase. The second is the semantic phase which associates, in some way, a meaning with each of the various subconstructs possible in the language. Meaning in this sense can be defined as the translation activity necessary whenever a specific subconstruct is recognized. The third phase is the actual generation of executable code for the machine specified.

This third phase is almost impossible to automate at this time due to the lack of commonality between the instruction sets and other hardware features of various different machines. The best existing approach is for the semantic phase of compilation to generate some sort of formal intermediate language which is then processed by a code generator handwritten for a specific machine.

The semantic phase is also difficult to automate at present. The main reason for this is the failure, to date, to develop a good formalization of semantic specifications. Some efforts have been made in this direction, notably by Lewis and Stearns [1], Feldman [2] and Lucas and Walk [3]. These, however, still fall short of what is necessary for complete automation of the semantics process. The most common approach to semantic processing is that used in the TWS to be described here. That is to transfer control, at points in the parsing process specified by the language designer, to hand coded routines which accomplish the semantic functions.

The algorithm which converts context free (CF) productions to modified Floyd Production Language (FPL) productions, which is described here, was developed to automate the parsing phase of compiler generation. It grew from the desire to implement a practical and efficient TWS. The goals of this TWS are the same as those of any TWS. The most important of these is the building of compilers which are fast, occupy as little space as possible and are capable of generating efficient object code. Input to a TWS should be in as simple a form as possible. A TWS should also be capable of processing the grammars of a large class of languages.

Out of the broad class of problems associated with translator writing systems, this thesis concentrates on one, namely the generation of parsing algorithms. Consideration was given to several different parsing algorithms as potential bases for the system. Among these were the algorithms of McKeeman [4], Wirth and Weber [5], Ingerman [6], Brooker and Morris [7] and Trout [8]. Without precluding subsequent incorporation of some parts of these methods, it was decided that some version of the production language (FPL) of Floyd [9], Evans [10] and Feldman [2] was potentially most likely to meet the first of the above criteria. The input to such a system is, however, more complex than was desirable.

Context free (CF) productions were chosen as the most widely known and easily understood method of language syntax specifications. The most commonly used notation for this type of syntactic specification is the Backus-Naur Form (BNF) as used in the ALGØL 60 report [11]. Attempts at conversion of CF grammars to FPL by Earley [12] and DeRemer [13] were considered and that of DeRemer chosen as the more promising starting point.

No attempt is actually made to generate parsers for all CF grammars. The goal, instead, is to generate parsers for a class of grammars which approaches, as nearly as possible, the LR(k) grammars defined by Knuth [14]. This restriction is made for two reasons. First, it is the belief here that attempts to process a larger subset of CF grammars are unlikely to achieve practical results; and second, it is felt that, if a designer set out to design an unambiguous grammar to specify the structural properties of a programming language, his result will be a grammar whose sentences can be parsed during a single, deterministic scan from left to right; i.e., an LR(k) grammar. Intuitively, we feel that this situation exists because the language is meant to be written and read by humans, most of whom are used to reading and writing natural languages from left to right.

This thesis consists of two parts. The first, chapters 2 through 5, is a precise description of an algorithm for the conversion of a set of CF productions to a set of deterministic FPL productions. The second, chapters 6 through 8, is an evaluation of this algorithm. Chapter 6 attempts to identify and classify that subset of CF grammars which the algorithm can successfully convert. Chapter 7 compares the resulting parsing algorithms to those generated by the recent somewhat similar systems of DeRemer [15], Korenjak [16] and Earley [17]. Chapter 8 considers the effectiveness of the algorithm as a part of an implemented TWS.

2. NOTATION AND DEFINITIONS

The notation and definitions of this chapter will be used throughout the remainder of this work. All notation and definitions used in the following text are not included in this chapter since some would be meaningless if not introduced in the proper context.

We consider a language L to be defined by a context free (CF) phrase structure grammar

$$G = (V_T, V_N, S, P)$$

where

V_T = the set of terminal symbols of L, which will be represented by lower case Latin letters,

V_N = the set of nonterminal symbols, which will be represented by upper case Latin letters from the beginning and middle of the alphabet,

$S \in V_N$ is a designated nonterminal symbol called the objective symbol and

P is a numbered set of rules, or productions, of the form $A \rightarrow \alpha$, where $A \in V_N$ and $\alpha \in (V_N \cup V_T)^*$.

Such a CF grammar is automatically extended as follows:

$\#$, an end of string marker, is added to V_T,

Σ is added to V_N and replaces S as the objective symbol and

$\Sigma \rightarrow \#S\,\#^k$ is added to P as production number zero.

A Symbols which may be either terminal or nonterminal will be denoted by upper case Latin letters from the end of the alphabet. Strings of symbols will be denoted by Greek letters. A string which consists of a single symbol X repeated n times will be denoted X^n. The input string to be parsed is assumed to be of the form

$$\sigma \#^{k+1}$$

where σ is made up of terminal symbols only.

A symbol X ($X \in V = V_N \cup V_T$) is called a headsymbol of a nonterminal A if

$$A = X, \text{ or}$$
$$A \rightarrow X \ldots, \text{ or}$$

there exists a subset of P of the form

$$\begin{aligned} &A \rightarrow A_1 \ldots \\ &A_1 \rightarrow A_2 \ldots \\ &\quad\vdots \\ &A_n \rightarrow X \ldots \qquad\qquad n \geq 1. \end{aligned}$$

Analogously, a symbol $X \in V$ is called a tailsymbol of a nonterminal A if

$$A = X, \text{ or}$$

$$A \rightarrow \ldots X, \text{ or}$$

there exists a subset of P of the form

$$\begin{array}{ll} A \rightarrow \ldots A_1 & \\ A_1 \ldots A_2 & \\ \vdots & \\ A_n \rightarrow \ldots X & n \geq 1. \end{array}$$

An explicitly left recursive occurrence of a nonterminal symbol A is that occurrence as the first symbol on the righthand side (RHS) of a CF production of the form

$$A \rightarrow A \ldots$$

An implicitly left recursive occurrence of A is an occurrence of A as the first symbol on a RHS in a subset of P of the form

$$\begin{array}{ll} A \rightarrow A_1 \ldots & \\ A_1 \rightarrow A_2 \ldots & \\ \vdots & \\ A_n \rightarrow A \ldots & n \geq 1 \end{array}$$

A left recursive occurrence, either explicit or implicit, is defined analogously.

A descriptor is associated with each specific occurrence of a symbol X on the RHS of a CF production. The descriptor (π, n) represents the occurrence of X as the n^{th} symbol on the RHS of the CF production numbered π.

The context free productions (rules of P) will be referred to simply as productions, or for clarity, CF productions. The FPL statements will be referred to as productions (when the context makes clear which type of production is meant), or FPL productions.

An FPL production will be written in the form

$$\text{L1:} \quad X \left| \begin{array}{l} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{array} \right. \quad Q \,| \quad * \quad L$$

The label L1 may or may not be present. Most parsing algorithms make use of a parsing stack into which input symbols are pushed and in which reductions are made. For a set of FPL productions which is generated by this CF to FPL coversion algorithm, this stack consists of only a single symbol, called a parser symbol, or p-symbol.

The X in the FPL production above, if present, is a symbol which is to be compared to this p-symbol. If the p-symbol is not X, processing is transferred to the beginning of the next production in sequence.

The α_i are strings of terminal symbols which are referred to as lookahead strings. If no lookahead string matches the next symbols in the input stream, processing is transferred to the beginning of the next production in sequence.

The Q, if present, represents one of two parsing activities. It takes the form $\leftarrow Z$ when one of three types of marker symbols is to be pushed into a special marker stack. It takes the form $\rightarrow A(n)$ when the p-symbol is to be set to A and the top n ($n \geq 0$) symbols are to be popped from the marker stack.

The * may or may not appear and, if present, indicates that the next input symbol is to replace the parser symbol. This is referred to as a scan.

The L is either the label of the next production to be used or the special symbol DNT-A (to be explained in detail later) which indicates that the nonterminal A has just been recognized and the marker stack must be used to calculate dynamically the label of the next production to be used.

As will be seen in chapter 3, there is a specific FPL production for each descriptor (π, n). Thus there is a one to one correspondence between a specific occurrence of a symbol on the RHS of a CF production, a descriptor and a particular FPL production. In certain portions of the text which follows, these terms will be used somewhat interchangeably.

3. THE BASIC ALGORITHM

Intuitively, the basis of the algorithm is that once a particular symbol X has been recognized and found to correspond to that occurrence of X as the n^{th} symbol on the RHS of the CF production numbered π, one of three situations must exist. The first possibility is that the $n + 1^{st}$ symbol on RHS π is a terminal symbol, t. In this case, the next symbol is scanned and control is transferred to an FPL production which checks for t and takes appropriate action if it is found. The second possibility is that the $n + 1^{st}$ symbol on RHS π is a nonterminal symbol, A. In this case, a marker symbol corresponding to this specific occurrence of A is pushed into the marker stack, the next input symbol is scanned and control is transferred to the first of a group of FPL productions which tests for all possible terminal beginnings (head symbols) of A. The last possibility is that this X is the last symbol of an n-symbol RHS which defines a nonterminal A. In this case, the parser symbol is set to A and an appropriate number of marker symbols are popped from the marker stack. If the top marker symbol is for A, control is transferred to the start of a group of FPL productions which includes tests for all left recursive occurrences of A and a test for the specific occurrence of A which caused the marker to be pushed. If the top marker symbol is for B ($\neq$A) then control is transferred to the start of a group of FPL productions which tests for all possible nonterminal beginnings (head symbols) of B, except for left recursive occurrences of B itself.

What follows is a formal description of these intuitive ideas.

3.1 Label (Group) Determination

The FPL productions are grouped and a label is attached to the first production of each group. The labels are thus associated with entire groups of productions rather than with single productions.

The first step in the algorithm is to determine the labels of all of the FPL production groups which might be required in the course of a parse. Let $X_n(\pi)$ represent the n^{th} symbol on the RHS of the CF production numbered π. There are four rules for determining which labels (groups) must be created.

For each $A \in V_N$:

label TH-A (Terminal Heads) exists if $\exists \pi, n$ such that $A = X_n(\pi)$ and $n \geq 2$. 3.1.1

This group would be required when the first n-1 symbols of the CF production numbered π had been recognized causing the need to initialize a search for A.

For each $A \in V_N$:

label NTH-A (Nonterminal Heads) exists if $\exists \pi, n$ such that $A = X_n(\pi)$ and $n \geq 2$. 3.1.2

This group would be required when the top symbol on the marker stack indicates that a search for A is in progress and a nonterminal symbol other than A has just been recognized.

For each $A \in V_N$:

label NT (π, n) (Nonterminal at (π,n)) exists 3.1.3

for each π, n such that $A = X_n(\pi)$ and $n \geq 2$.

This group would be required when the top symbol on the marker stack indicates that a search for the occurrence of A as the n^{th} symbol on the RHS of the CF production numbered π is in progress and the non-terminal A has just been recognized.

For each $t \in V_T$:

label T (π, n) (Terminal at (π, n)) exists 3.1.4

for each π, n such that $t = X_n(\pi)$ and $n \geq 2$.

This group would be required when the first n-1 symbols of the CF production numbered π had been recognized and a test for t must be next.

3.2 Descriptor Set Generation

The next step in the CF productions to FPL productions conversion algorithm is to generate the set of descriptors, $D_Q = \{(\pi_i, n_i) \mid i = 1,2,\ldots,m\}$ for which corresponding FPL productions must be created for each FPL group Q. The group labels are descriptive in the sense that the four types of labels (TH, NTH, NT and T) generate descriptor sets in the following four different ways:

$$D_{TH\text{-}A} = \{ (\pi,1) \mid X_1(\pi) \in V_T \text{ and the LHS of } \pi \text{ is a head symbol of } A\} \quad 3.2.1$$

$$D_{NTH-A} = \{(\pi,1) \mid X_1(\pi) \in V_N, X_1(\pi) \neq A \text{ and the LHS of } \pi \text{ is a head symbol of } A\} \quad 3.2.2$$

$$D_{NT(\pi,n)} = \{(\pi',n') \mid (\pi,n) = (\pi',n') \text{ or } (\pi',n') \text{ describes a left recursive occurrence of } X_n(\pi)\} \quad 3.2.3$$

$$D_{T(\pi,n)} = \{(\pi,n)\} \quad 3.2.4$$

3.3 Descriptor to FPL Production Mapping

For each group, the descriptors are mapped on a one to one basis into FPL productions and the appropriate group label is attached to the first such production. As was suggested at the beginning of this chapter, there are three different mapping rules which could apply to the descriptor (π, n), depending on what, if anything, occurs as the $n + 1^{st}$ symbol on the RHS of the CF production numbered π.

Assume α is a string of symbols of length n-1. The three mapping rules are:

If symbol n + 1 is a nonterminal (i.e. 3.3.1

π is $A \rightarrow \alpha$ X B ...) then

(π, n) maps into

$X \mid \leftarrow \overline{B}\ (\pi,\ n+1) \mid \ *\ TH - B.$

If symbol n + 1 is a terminal (i.e. 3.3.2

π is $A \rightarrow \alpha$ X t ...) then

(π, n) maps into

$X \mid \qquad\qquad \mid \ *\ T(\pi,\ n+1).$

If CF production π has only n RHS symbols (i.e. π is $A \to \alpha X$) then (π, n) maps into 3.3.3

$X \mid \quad \to A\ (k) \quad \mid \quad$ DNT-A

where k is the number of nonterminal symbols in αX other than the first symbol and $\overline{B}\ (\pi', n')$ at the top of the marker stack implies the following definition of DNT-A.

If $A = B$, DNT-A = NT(π', n') else 3.3.4

if $A \neq B$, DNT-A = NTH - B.

All of the FPL productions in an NT (π, n) group have the same parser symbol field (see 3.2.3). Transfer is made to an NT (π, n) group only when the parser symbol has just been set to this particular nonterminal symbol (see 3.3.4). Therefore, when the above mapping rules are applied to the descriptors for an NT (π, n) group, the parser symbol field should be left empty.

In addition, two special labelled productions must be generated. They are

START: $\mid \quad \leftarrow \overline{\Sigma}\ (0,0), \ \leftarrow \overline{S}\ (0,2) \quad \mid$ * TH-S and

NT (0,0): $\mid \quad\quad \mid$ EXIT.

3.4 Preclusion and Error Detection

If, within any group of FPL productions, the parser symbol of one production is the same as the parser symbol of another, the first in sequence precludes any possibility that the second could be successfully applied. Methods for the elimination of such preclusion are the subject of chapters 4 and 5.

Each group is designed to contain all possible correct alternatives at the point reached in the parse upon entry to the group. Thus an error in the input stream would cause entry to some group in which no production would apply and control would go to the first production after the group. For this reason an error production is inserted at the end of each group.

3.5 Summary and Example

The algorithm consists of the following five steps:

1. Read the CF grammar of L and add the special 0-th production $\Sigma \rightarrow \#S\ \#^k$;
2. Determine the labels of the FPL groups required (3.1);
3. Generate descriptor sets for each group (3.2);
4. Map the descrptors into FPL productions and create the special productions START and NT (0,0);
5. Do the necessary preclusion elimination and add the FPL error production to each group.

The following example has been chosen to contain one preclusion and illustrate rules one through four. It is assumed in this example that each group is followed by an error production and k has a value of one.

3.5.1 Input Syntax

CF Production Number	LHS		Position 1	2	3
0	Σ	→	#	S	#
1	S	→	A	B	c
2	A	→	a		
3	A	→	D		
4	B	→	b		
5	B	→	B	b	
6	D	→	d		

3.5.2 Labels (Groups) Needed

TH - S	NT (0,2)	T (5,2)
TH - B	NT (1,2)	START
NTH - S	T (0,3)	NT (0,0)
NTH - B	T (1,3)	

3.5.3 Descriptor Sets

$$D_{TH-S} = \{(2,1), (6,1)\}$$

$$D_{TH-B} = \{(4,1)\}$$

$$D_{NTH-S} = \{(1,1), (3,1)\}$$

$$D_{NTH-B} = \{\quad\}$$

$D_{NT(0,2)}$	=	$\{(0,2)\}$
$D_{NT(1,2)}$	=	$\{(1,2), (5,1)\}$
$D_{T(0,3)}$	=	$\{(0,3)\}$
$D_{T(1,3)}$	=	$\{(1,3)\}$
$D_{T(5,2)}$	=	$\{(5,2)\}$
D_{START}		special
$D_{NT(0,0)}$		special

3.5.4 <u>Resulting FPL Production Set</u>

START:		$\leftarrow \overline{\Sigma}\ (0,0), \leftarrow \overline{S}\ (0,2)$		*	TH-S
$NT_{(0,0)}$:					EXIT
TH-S:	a		$\rightarrow A_{(0)}$		DNT-A
	d		$\rightarrow D_{(0)}$		DNT-D
TH-B:	b		$\rightarrow B_{(0)}$		DNT-B
NTH-S:	A		$\leftarrow \overline{B}_{(1,2)}$	*	TH-B
	D		$\rightarrow A_{(0)}$		DNT-A
NTH-B:					
$NT_{(0,2)}$:				*	$T_{(0,3)}$
$NT_{(1,2)}$:				*	$T_{(1,3)}$
				*	$T_{(5,2)}$ preclusion
$T_{(0,3)}$	#		$\rightarrow \Sigma_{(1)}$		DNT-Σ
$T_{(1,3)}$	c		$\rightarrow S_{(1)}$		DNT-S
$T_{(5,2)}$	b		$\rightarrow B_{(0)}$		DNT-B

4. COMBINATION OF FPL PRODUCTIONS

Efficient elimination of preclusions is the key to the practical success of this conversion algorithm since the class of grammars which do not generate preclusions is extremely small. A left recursive occurrence, for example, of any nonterminal symbol causes a preclusion in all NT (π, n) groups for that symbol. It is, in fact, the failure to eliminate a preclusion which defines a CF grammar as unacceptable to the conversion algorithm. Thus, the more effective the preclusion elimination, the larger will be the class of convertible grammars.

There are two more or less independent methods of eliminating preclusion. The first of these is the combination of two or more conflicting FPL productions into a single production. The second is the generation of strings of lookahead symbols for each of the conflicting productions. The order in which these two methods are applied makes almost no difference. Due, at least in part, to the details of our particular implementation, it was found that the most effective and efficient approach is to try differentiation by one symbol lookahead first. If this failed, combination is attempted. As a last resort, a k symbol lookahead is attempted.

One feature of the conversion algorithm, used to eliminate preclusion, is the combination of two or more FPL productions in a group into a single production. Two (or more) productions are combined when the first in sequence precludes the second (the rest). Each production which is formed by such a combination, creates the need for one or more additional groups. As will be seen, these new

group types are, in each case, unions of those groups which would otherwise be transferred to either directly or indirectly (through a later application of the dynamic DNT transfer) by the simple productions being combined.

4.1 CT(m) Groups

The simplest combination possible is of two or more FPL productions formed by mapping rule 3.3.2 (when the next RHS symbols are terminals). If, within a group, the productions

$$X \quad | \quad\quad | \quad * \quad T(\pi_1, n_1)$$

$$\vdots$$

$$X \quad | \quad\quad | \quad * \quad T(\pi_k, n_k)$$

should appear, and any preceding attempt at differentiation by look-ahead has failed, then they can be combined into the single production

$$X \quad | \quad\quad | \quad * \quad CT(m)$$

where this is the m^{th} combination to have been made.

Associated with the integer m is the set of labels

$$L_m = \{T(\pi_i, n_i) \mid i = 1,2,\ldots, k\} \qquad 4.1.1$$

The descriptor set $D_{CT(m)}$ is then defined by

$$D_{CT(m)} = \bigcup_{q \in L_m} D_q . \qquad 4.1.2$$

Another, more intuitive, definition of $D_{CT(m)}$ is

$$D_{CT(m)} = \{(\pi, n) \mid (\pi, n-1) \text{ is descriptor of one of the productions being combined}\}.$$

Group CT(m) is, thus, precisely the union of the various groups which could have been transferred to, had the combination not been made.

Some of the productions of the CT(m) group could have the same parser symbol. In fact, if lookahead has been previously attempted prior to the use of combination, a CT(m) group will always be of the form

$$\begin{array}{lll} CT(m): & t & | \quad \ldots \\ & & \vdots \\ & t & | \quad \ldots \end{array}$$

As is easily seen, the preclusion has not been eliminated, but simply moved to a different group. This group, however, is entered at a point one symbol farther into the input stream and therefore, the maximum length lookahead string reaches one symbol farther into the input. The net effect of the combination in this case, is the implicit one symbol extension of the maximum lookahead capability of the conversion algorithm.

4.2 Marker Symbol Combination

The second class of combinations is of two or more FPL productions formed by mapping rule 3.3.1 (when the next RHS symbols are nonterminals). This is referred to as a class of combinations because there are two different types of combinations which can be made in this circumstance. These two types of combination can be made in any order and, in fact, are made in a relatively random sequence in the actual implementation of the algorithm. It is, however, simpler to describe and easier to understand if it is assumed that all possible combinations of the first type are made before any combinations of the second type.

The first type of combination is possible when the marker symbols (corresponding to the next RHS symbols) to be pushed into the marker stack are for different occurrences of the same nonterminal symbol. If, within a group, the productions

$$X \mid \leftarrow \bar{A}\,(\pi_1, n_1) \mid \quad * \quad \text{TH-A}$$

$$\vdots$$

$$X \mid \leftarrow \bar{A}\,(\pi_k, n_k) \mid \quad * \quad \text{TH-A}$$

should appear, and any preceding attempt at differentiation by look-ahead has failed, then they can be combined into the single production

$$X \mid \leftarrow \bar{A}*\,(m) \mid \quad * \quad \text{TH-A}$$

where this is the m^{th} combination to have been made. The symbol $\bar{A}*$ (m)

will henceforth be referred to as a special marker symbol (as opposed to a simple marker symbol $\bar{A}$ (π, n)). The possibility that the symbol at the top of the marker stack is a special marker symbol requires an extension of the definition of the dynamic transfer DNT (3.3.4). It now is, if the top marker symbol is $\bar{B}$ (π, n) then 4.2.1

$$\text{if } A = B\ ,\quad \text{DNT-A} \;=\; \text{NT } (\pi,\ n) \text{ else}$$

$$\text{if } A \neq B\ ,\quad \text{DNT-A} \;=\; \text{NTH-B or}$$

if the top marker symbol is $\bar{B}*$ (m) then

$$\text{if } A = B\ ,\quad \text{DNT-A} \;=\; \text{CNT } (m) \text{ else}$$

$$\text{if } A \neq B\ ,\quad \text{DNT-A} \;=\; \text{NTH-B.}$$

As can be seen, the label of the group which must be built whenever a combination of this type is made, is CNT (m). The symbol $\bar{A}*$ (m) actually represents the set of marker symbols which could be pushed by the individual FPL productions which were combined. That is,

$$\bar{A}* \ (m) \;=\; \{\bar{A}\ (\pi_i,\ n_i) \ \mid\ i = 1,2,\ldots,\ k\}.$$

The descriptor set $D_{CNT(m)}$ is then defined by

$$D_{CNT(m)} = \{(\pi,\ n) \mid \bar{X}\ (\pi',\ n') \in \bar{X}*\ (m) \text{ and } (\pi,\ n) = (\pi',\ n') \text{ or } (\pi,\ n) \text{ describes a left recursive occurrence of } X\}. \qquad 4.2.2$$

As can be seen by comparing this definition of $D_{CNT}(m)$ with the definition of $D_{NT}(\pi, n)$ (3.2.3), this type of group is simply the union of those groups which would have been needed had no combination been made.

The second possible combination of FPL productions formed by mapping rule 3.3.1 is of those productions which cause marker symbols (simple or special) of different nonterminal symbols to be pushed into the marker stack. This cannot be done under a certain condition, but it will be shown later in this chapter how this condition can be eliminated. As was mentioned earlier, it is assumed that all combinations which create special marker symbols, have been made.

If, within a group, the productions

$$X \mid \leftarrow \overline{Q}_1 \mid * \quad \text{TH-A}_1$$

$$\vdots$$

$$X \mid \leftarrow \overline{Q}_k \mid * \quad \text{TH-A}_k$$

should appear with $\overline{Q}_i = \overline{A}_i\ (\pi_i, n_i)$ or $\overline{Q}_i = \overline{A}^*_i\ (m_i)$, and any preceding attempt at differentiation by lookahead has failed, then they can be combined into the single production

$$X \mid \leftarrow (\overline{m}) \mid * \quad \text{CTH}\ (m)$$

where this is the m^{th} conbination to have been made. The symbol $(\overline{m})$ is called a combined marker symbol and represents the set of marker

symbols which could have been pushed into the marker stack by the various productions which were combined. That is:

$$(\overline{m}) = \{\overline{Q}_1, \overline{Q}_2, \ldots, \overline{Q}_k\}.$$

The sets represented by the combined marker symbols must be available when the FPL production set is used to parse an input string because they are used in the dynamic transfer DNT. The definition of DNT is extended to read

if the top marker symbol is $\overline{B}$ (π, n) then 4.2.3

if $A = B$, DNT-A = NT (π, n) else

if $A \neq B$, DNT-A = NTH-B or

if the top marker symbol is $\overline{B}*$ (m) then

if $A = B$, DNT-A = CNT (m) else

if $A \neq B$, DNT-A = NTH-B or

if the top marker symbol is $(\overline{m})$ then

if $\overline{A}$ $(\pi, n) \in (\overline{m})$, DNT-A = NT (π, n) else

if $\overline{A}*$ $(m') \in (\overline{m})$, DNT-A = CNT (m') else

if $\overline{A}$ $(\pi, n) \notin (\overline{m})$ and $\overline{A}*$ (m) $\notin (\overline{m})$,

DNT-A = CNTH (m).

The production formed by a combination of this type indicates a transfer to a group labelled CTH (m). This type of group is necessary whenever such a combination is made. Its descriptor set $D_{CTH(m)}$ is indirectly defined by $(\overline{m})$ as follows:

$$D_{CTH(m)} = \bigcup_{\substack{\overline{A}(\pi,\ n)\ \epsilon\ (\overline{m}) \\ \overline{A}*\ (m')\ \epsilon\ (\overline{m})}} D_{TH\text{-}A} \qquad 4.2.4$$

The dynamic transfer definition indicates a potential transfer to a group labelled $CNTH_{(m)}$. This type of group is also necessary whenever such a combination is made. Its descriptor set $D_{CNTH(m)}$ is indirectly defined by $(\overline{m})$ as follows:

$$D_{CNTH(m)} = \bigcup_{\substack{\overline{A}(\pi,\ n)\ \epsilon\ (\overline{m}) \\ \overline{A}*(m')\ \epsilon\ (\overline{m})}} D_{NTH\text{-}A} \qquad 4.2.5$$

4.3 Initial CF Grammar Revision

Section 4.1.2 made reference to a condition which prevented the combination of productions created by mapping rule 3.3.1 when the marker symbols (simple or special) to be pushed are for difference non-terminal symbols. This occurs when the nonterminal symbols associated with the marker symbols to be pushed into the marker stack are such that one is a head symbol of another. That is, when $\overline{A}\ (\pi_i,\ n_i)$ (or $\overline{A}*\ (m_i)$) and $\overline{B}\ (\pi_j,\ n_j)$ (or $\overline{B}*\ (m_j)$) would be elements of $(\overline{m})$, $A \neq B$ and A is a head symbol of B. When $(\overline{m})$ was atop the marker stack, this would always cause DNT-A to be equal NT $(\pi_i,\ n_i)$ (or CNT (m_i)). This ignores completely the possibility that the particular A just recognized is the beginning of a derivative of B (in which case, DNT-A should equal CNTH (m)).

This problem can be overcome by a relatively simple revision of the CF grammar before any CF to FPL conversion is done. This involves a search of all the CF productions for pairs of the form

$$C \rightarrow \alpha B \ldots$$
$$D \rightarrow \alpha X \quad \beta$$

or
$$C \rightarrow A' \gamma \; B \ldots$$
$$D \rightarrow \alpha A \gamma \; X \; \beta$$

or
$$C \rightarrow \alpha A \gamma \; B \ldots$$
$$D \rightarrow A' \gamma X \; \beta$$

where C may or may not be the same as D, α is a nonempty string, β and γ are possibly empty strings, A' is a left recursive occurrence of A and $X \in V_N$ is a head symbol of B. For each such pair, $X\beta$ is replaced by a new nonterminal symbol F, and the production

$$F \rightarrow X \beta$$

is added to the set of CF productions. This revision of the CF grammar does not change the language being specified but does keep the previously described condition from occurring and preventing the combination of productions when necessary.

It can be noted at this point that any FPL productions formed by mapping rule 3.3.1 may be combined and that any FPL produc-

tions formed by mapping rule 3.3.2 may be combined. It is not possible, however, to combine two productions when one is formed by rule 3.3.1 and the other by rule 3.3.2. The possibility that such a pair of productions might appear within a group and require combination can be completely eliminated by one of two simple extensions of the CF grammar revision described above. The first case is when a differentiation of precluding FPL productions by lookahead is to be attempted before any conbination is attempted. In this case the CF grammar revision is the same except that it is also applied when X is a terminal symbol. When no attempt at lookahead differentiation is to precede possible combination attempts, the CF grammar revision must be further extended to include all those cases where X is a terminal symbol, even if it is not a head symbol of B.

Thus, the only case where combination of FPL productions cannot be used to overcome preclusion is when one or more of the FPL productions involved has been generated by mapping rule 3.3.3 (i.e. when the end of RHS has been reached and a reduction is necessary). Intuitively, this is explained by the fact that combination of FPL productions is simply a deferring of decision making which cannot be done when one of the possible decisions is to perform a specific parsing activity (i.e. reduction of a definition to its LHS).

4.4 Summary and Example

The basic conversion algorithm (chapter 3) is now extended to include an initial CF grammar revision. In addition, preclusions which have not been resolved by a preceding application of lookahead and do not involve an FPL production which is generated by

mapping rule 3.3.3 may be eliminated by combining the FPL productions involved. Each such combination creates the need for one or more new groups. These new groups are, in each case, exactly the union of appropriate groups which would have been necessary had no combination been made. These combination procedures use the new special marker ($\bar{A}$* (m)) and combined marker (($\bar{m}$)) symbols and the definition of the dynamic transfer, DNT, is extended (4.2.3) to include the cases where these new marker symbols are at the top of the marker stack.

The example which follows has been chosen to illustrate the ideas of this chapter. Some segments of the resulting FPL production set have been omitted since they do not illustrate material from this chapter.

4.4.1 <u>Input Syntax</u>

CF Production Number	LHS		Position 1	2	3
0	Σ	→	#	S	#
1	S	→	a	A	f
2	S	→	a	B	c
3	S	→	a	d	d
4	S	→	a	B	d
5	S	→	b	c	
6	S	→	b	d	
7	B	→	A	b	
8	B	→	b		
9	A	→	a		

4.4.2 Grammar Revisions

Production 1 becomes

1	S	→	a	D_1	and
10	D_1	→	A	f	

is added.

Production 3 becomes

3	S	→	a	D_2	and
11	D_2	→	d	d	

is added.

4.4.3 FPL Production Subset

TH-S:	a	← $(\bar{2})$	*	CTH (2)
	b		*	CT (3)
CTH (2):	b	→ B(0)		DNT-B
	a	→ A(0)		DNT-A
	d		*	T (11,2)
CNTH (2):	A		*	CT (4)
CNT (1):	B		*	CT (5)

CT (3):	c	$\mid\rightarrow S(0) \mid$	DNT-S
	d	$\mid\rightarrow S(0) \mid$	DNT-S
CT (4)	b	$\mid\rightarrow B(0) \mid$	DNT-B
	f	$\mid\rightarrow D_1(0) \mid$	DNT-D_1
CT (5)	c	$\mid\rightarrow S(1) \mid$	DNT-S
	d	$\mid\rightarrow S(1) \mid$	DNT-S

4.4.4 <u>Combinations</u>

$$\bar{B}^* (1) = \{ \bar{B}\ (2,2)\ ,\ \bar{B}\ (4,2)\}$$

$$(\bar{2}) = \{ \bar{D}_1\ (1,2)\ ,\ \bar{B}^*\ (1)\ ,\ \bar{D}_2\ (3,2)\}$$

$$L_3 = \{ T\ (5,2)\ ,\ T\ (6,2)\}$$

$$L_4 = \{ T\ (7,2)\ ,\ T\ (10,2)\}$$

$$L_5 = \{ T\ (1,3)\ ,\ T\ (3,3)\}$$

5. LOOKAHEAD CONSTRUCTION

As was pointed out in chapter 4, there are two relatively independent methods of preclusion elimination. The first of these, combination of FPL productions, was described in chapter 4. The second is explicit use of the next symbols in the input stream to see if they correspond to one of a set of strings of symbols which could follow if a particular FPL production should apply.

If the parser symbols of two or more FPL productions within a group are identical, then a preclusion exists and combination of productions or lookahead must be used to differentiate. It will be assumed throughout this chapter that all possible combinations have been made. When combinations have not already been made, lookahead construction is a simple case of the general method to be described. The method of lookahead construction is most easily described by discussing explicitly the implementation used.

The first production in the sequence of productions involved in the preclusion is paired, in turn, with each of those productions which it precludes. After the lookahead for this first production has been completely constructed (and assuming success in differentiation), it is, in effect, removed from the set of productions involved in the preclusion, thus reducing the set by one. This new smaller set is then processed by pairing its first production with all those which it precludes. This process continues until only one production remains. This last production precludes nothing and, therefore, needs no lookahead. This approach guarantees that each production has only as much lookahead as it needs to avoid precluding those productions which follow it in the group.

Two identical lookahead construction buffers are used; one for each of the FPL production in a pair. Each buffer is initialized when it is first assigned to a production. For a given set of precluding productions, the first buffer is thus initialized once from the first production, and the second buffer is reinitialized each time this first production is paired with another production from among those which it precludes. Assuming the productions involved in the preclusion are numbered in sequence from one to n, the flow chart of Figure 1 describes this pairing and initialization procedure.

Each line of these two buffers has two words of control information plus k additional words, where k is the length of the lookahead presently being generated. Associated with each FPL production is a set of descriptors, one for each of the productions combined to form this resulting FPL production. One line, starting with line one, is initialized for each such descriptor, (π, n), when a lookahead buffer is initialized from an FPL production. This is done as follows:

word 1: the nonterminal which is the LHS of the CF production numbered π

word 2: zero

words 3 thru x: RHS symbols $n + 1$ through $n + y$ from CF production π

words $x + 1$ thru $k + 2$: zero

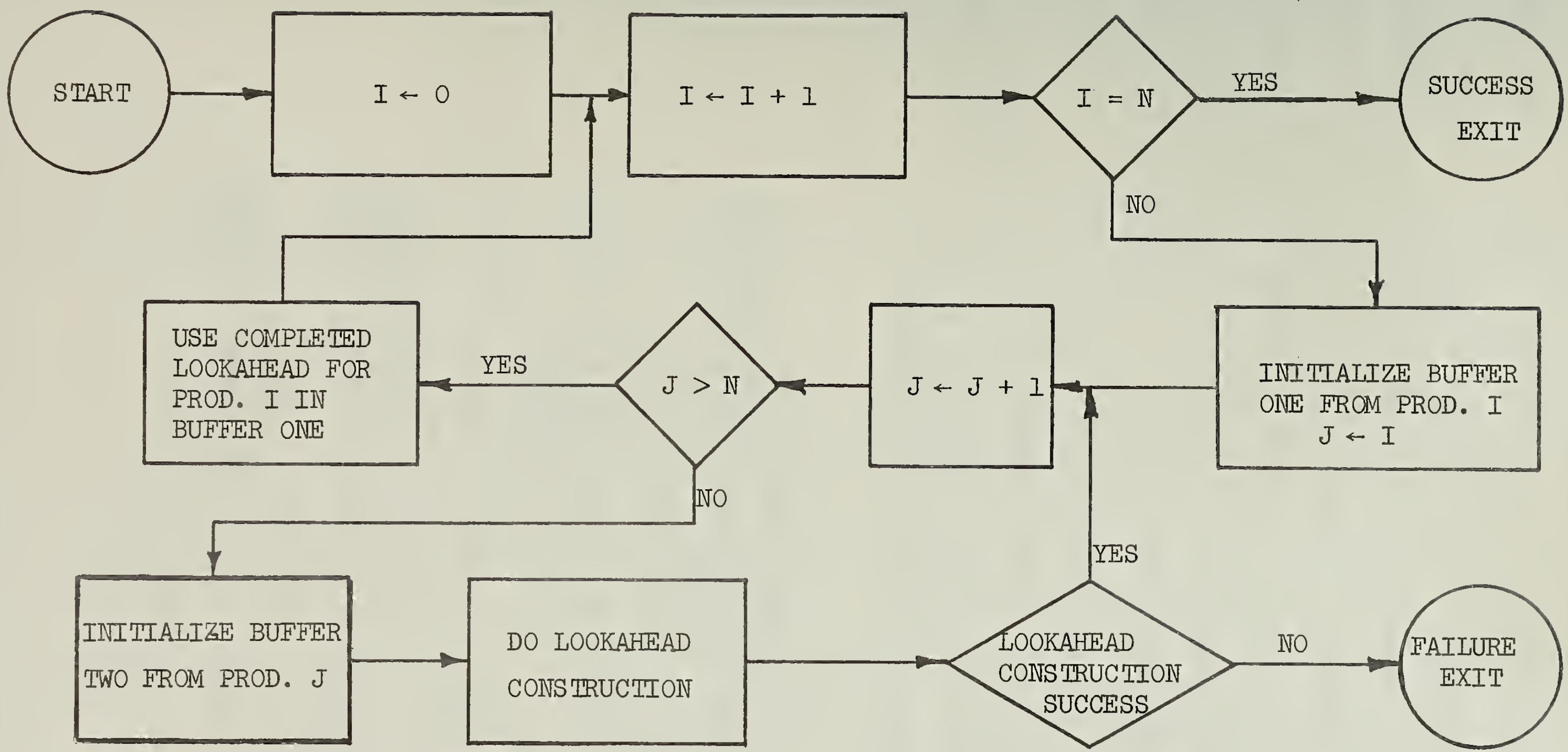


Figure 1. Flow Chart for Lookahead Buffer Usage

where m + n is the number of RHS symbols of the CF production numbered π, x is minimum (k + 2, m + 2) and y is minimum (k, m). The effect of this line initialization is to fill the lookahead string field (words 2 through k + 2) as much as possible directly from the appropriate CF production (π). Word 1 is used (when necessary) to extend those strings which have not yet reached k symbols in length. In buffer one, word 2 is used to specify how many of the lookahead string symbols of a given line are terminals which are required to differentiate the corresponding FPL production from one (or more) of the FPL productions which it precludes. Word 2 in the lines of buffer two is meaningless.

Figure 2 is the flow chart for the generation of lookahead with the variables used defined as follows:

I, J, L are loop indices

LK1 is the buffer associated with the first FPL production

LK2 is the buffer associated with the second FPL production

K is the length of the lookahead being constructed

LK1PT is a pointer to the first empty line of LK1

LK2PT is a pointer to the first empty line of LK2

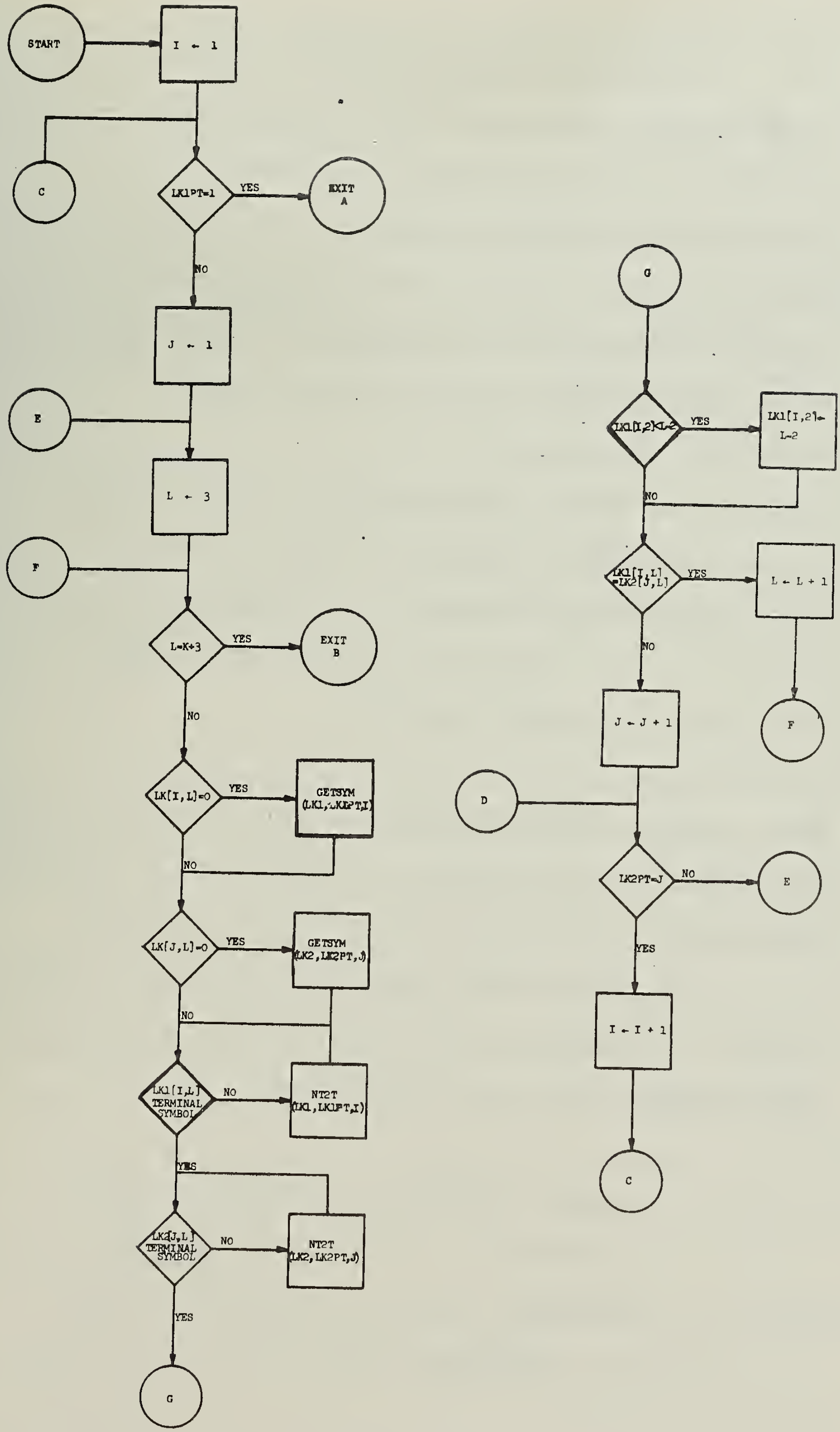


Figure 2. Flow Chart for Lookahead Generation

The procedure GETSYM extends the string in line I (J) of buffer LK1 (LK2) with pointer LK1PT (LK2PT) by finding, in the set of CF productions, each occurrence of a nonterminal symbol which is not the last symbol of a RHS, and which has the nonterminal symbol at LK1 [I, 1] as a tail symbol. Each such occurrence indicates a symbol or symbols which may follow the partial string already constructed and is used to create a new line in LK1 as follows:

> word 1: the LHS symbol of the occurrence
>
> words 2 through L-1: identical to row I
>
> words L through K + 2: the symbols which follow the occurrence in that RHS (left justified).

This, in effect, extends incomplete strings by adding all substrings which could possibly follow the nonterminal, one of whose definitions (RHS) was used to generate the FPL production originally.

The procedure NT2T expands the nonterminal at LK1 [I, L] (LK2 [J, L]) by replacing it with each RHS which has it as a LHS. Each such RHS creates a new line as follows:

> words 1 through L-1: identical to line I;
>
> starting at L: the symbols of the RHS
>
> remainder of line: the words of line I starting at word L + 1.

For both GETSYM (GET SYMbols) and NT2T (NonTerminal to Terminal conversion), line I is saved at the end of the buffer. Each new line created is compared with all lines previously processed (lines one through I-1 and the lines at the end of the buffer). If the new line (except for word 2) is identical to any of these, it is ignored. The first new line not ignored is put at line I. The rest are placed starting at LK1PT (which is incremented by one for each such line). If all new lines are ignored, the LK1PT is decremented by one, line LK1PT is moved to line I, and control is transferred to point C of Figure 1. If this occurs while processing line J of LK2, the same procedure is followed with LK2PT and control is transferred to point D. This activity assures termination of the iterative lookahead construction (see chapter 7).

Exit A is taken when sufficient lookahead has been generated for the first FPL production to prevent it from precluding the second. Each additional production precluded by this first production is expanded into LK2 and the process outlined by Figure 2 is repeated. The lookahead already generated may prove sufficient or an extension of it may be generated. If exit A is taken and no further preclusion exists, then lookahead generation is complete for this production. Each line of LK1 (from the beginning) represents a different lookahead string and word 2 indicates how many of the symbols are necessary. It should be noted that all lookahead strings are not of length K but rather the minimum length necessary.

Exit B is the failure exit and is taken when the two productions cannot be differentiated by a lookahead (of length K).

This method of lookahead construction makes no use whatsoever of the group in which an FPL production appears and thus can introduce some extraneous lookahead strings. Some other methods of extending strings (different GETSYM) using directed graphs to represent the language structure were worked on but all appeared impractical. It can also be pointed out that throughout the use of this system, no case has been noted where a more accurate lookahead construction would have differentiated precluding productions when the practical method described in this chapter had failed.

Those readers who are interested only in the practical and implemented TWS which uses this CF to FPL conversion algorithm as the basis of its syntax preprocessor should skip to chapter 8.

6. CF GRAMMARS ACCEPTED

This chapter will define a class of CF grammars which is a relatively large subset of all LR (k) CF grammars. It will then be shown that any CF grammar which falls within this newly defined class can be successfully converted to a deterministic set of FPL productions by the algorithm described in chapters 3 through 5.

6.1 Strong LR (k) Grammars

The RHS of any CF production, which has as its LHS the nonterminal symbol A, will be referred to as a definition of A. A derivative of a nonterminal A is recursively defined to be either a definition of A or the result of replacing the occurrence of any nonterminal B (possibly equal A) in a derivative of A by a definition of B. A derivative of A can also be defined as any string α, such that an appropriate sequence of replacements of definitions, which appear as substrings of α, by the nonterminal symbols which they define will ultimately result in A. This process of replacing a substring by the nonterminal which it defines is referred to as a reduction. Thus, a derivative of A is any string which can be reduced to A. A terminal derivative of A is a derivative which includes only terminal symbols.

We now define a right derivative of A to be either a definition of A or the result of replacing the last (i.e., rightmost) nonterminal symbol in a derivative of A by some definition of that nonterminal symbol. The set of right terminal derivatives of a nonterminal symbol A is correspondingly defined to be the subset of right

derivatives of A which contain only terminal symbols. While the set of right derivatives of A does not include all derivatives of A, the set of right terminal derivatives of A is clearly identical to the set of terminal derivatives of A.

Each terminal string α, which is in a language defined by CF grammar G (with Σ defining production zero included) must be a right terminal derivative of Σ. If G is unambiguous then, by definition, there must be a specific sequence of rightmost nonterminal symbol replacements for each such α. This sequence is referred to as the right to left (RL) generation, or derivation, of α from Σ. If this sequence is inverted, the definitions in α being replaced by the appropriate nonterminal symbols, it is referred to as the left to right (LR) reduction of α to Σ. Recognizing and making this LR reduction of α to Σ is precisely the goal of a left to right parse of any terminal string in the language defined by G.

In determining the next step in the LR reduction (parse) of a string α to Σ, two things must be determined. The first is the position n, of the end of the substring which must be replaced by the appropriate nonterminal symbol. The beginning of the substring would be of equal value but the end is used because the definition of the LR reduction process guarantees that the end cannot be to the left of the previous reduction and thus makes it somewhat easier to locate. The second fact which must be determined is which production (reduction) must be applied. This is partially determined by the symbol string preceding position n, but, in general, the RHS of more than one production could apply. $\beta \mid_{\pi} \gamma$ is defined to be the reducible form of a string α if and only if $\alpha = \beta\gamma$ and the next step in the LR reduction

of α to Σ is the replacement of the RHS of production π which ends the substring β by the LHS of production π. In the definitions which follow, $\beta = \beta'X$.

Several definitions exist for the LR (k) class of CF grammars. Among these are the original definition of Knuth [14] and the variations of DeRemer [15] and Hopcroft and Ullman [18]. Each of these is formulated slightly differently so as to serve the specific purposes of the respective authors. For the purposes of this work, the following equivalent definition is used:

> Let k be a non-negative integer. A CF grammar G is LR (k) if and only if every right derivative α of Σ has a unique reducible from $\beta'X \underset{\pi}{|} \gamma$ which can be determined by investigating only β', X and the first k symbols of γ.

Floyd [19] has defined a somewhat more restricted class of CF grammars which he calls left to right bounded context, LR (m, n), grammars. We define LR (m, n) grammars as follows:

> Let m and n be non-negative integers. A CF grammar G is LR (m, n) if and only if every right derivative α of Σ has a unique reducible form $\beta'X \underset{\pi}{|} \gamma$ which can be determined by investigating only X, the last m symbols of β' and the first n symbols of γ.

We now define strong LR (k) grammars, SLR (k) grammars, in a similar manner, as follows:

> Let k be a non-negative integer. A CF grammar G is SLR (k) if and only if every right derivative α of Σ has a reducible form $\beta' \; X \;|_{\pi} \; \gamma$ which can be determined by investigating only X and either β' <u>or</u> the first k symbols of γ.

Using these definitions it is easy to see that the class of LR (m, n) grammars is a subset of the class of LR (k) grammars (since an LR (k) grammar could be considered to be an LR (∞, k) grammar). It is also obvious from the definitions that the class of SLR (k) grammars is a subset of the class of LR (k) grammars since the definitions are identical except for the use of 'or' instead of 'and' in reference to the context which is used in making parsing decisions. No such clear cut relationship holds between the class of SLR (k) grammars and the class of LR (m, n) grammars. The CF grammar

1 A $\rightarrow$ a B c D

2 A $\rightarrow$ b B D f

3 B $\rightarrow$ x c

4 B $\rightarrow$ x

5 D $\rightarrow$ d D

6 D $\rightarrow$ d

is not SLR (k) for any k since both the x in production three and the x in production four can be preceded by either a or b. Both of these x's can be followed by cd^k. Thus, using only lookahead <u>or</u> only look-back one cannot decide whether x, when found (and followed by cd^k), should be reduced to B. This grammar is, however, LR (1, 2) since x in the context (a, cd) must be from production four and should be reduced to B while x in the context (b, cd) must from production three. The CF grammar

1	A	→	a B
2	A	→	b D
3	B	→	c B
4	B	→	d
5	D	→	c D
6	D	→	d

is not LR (m, n) for any finite m since the d in the input string $a\ c^m\ d$ or $b\ c^m\ d$ must be reduced to B (production four) or D (production six) depending on whether the first symbol of the string, which is m + 1 symbols back, is a or b. This grammar is, however, SLR(0) by definition.

An intuitive feeling for what is and what is not included in the SLR (k) class of CF grammars is difficult to obtain. That SLR (k) grammars are not all LR (k) grammars is clear from the first example above. It is the feeling of this author that the class of SLR (k) grammars includes almost all LR (k) grammars which might reasonably be expected from human specification of meaningful languages. This is based on the difficulty of generating the somewhat contrived example

above and, more significantly, on the fact that extensive use of the CF to FPL conversion algorithm, which will be shown to accept all SLR (k) grammars, has uncovered no example of an LR (k) grammar which is not SLR (k). SLR (k) grammars do include the "standard" syntax for expressions, block structures and other commonly occurring things of this nature.

The remainder of the chapter is a proof of the fact that the CF to FPL conversion algorithm of Chapters Three through Five can successfully convert all SLR (k) grammars. It is clear from the definitions of this chapter and the description of the algorithm that what needs proof is the adequacy of the investigation of the preceding or following context.

6.2 Explicit Lookahead Adequacy

This section will show that the lookahead construction method described in Chapter Five, generates all and only those terminal symbol strings which could possibly be next in the input stream if a particular CF RHS occurrence is being recognized and a particular FPL production is therefore to be the next step in a parse.

Some additional notation must be introduced at this point. For each descriptor (π_i, n_i) there are the following:

q_i is the number of RHS symbols of CF production π_i;

α_i is the string of symbols with descriptors $(\pi_i, n_i + 1), (\pi_i, n_i + 2), \ldots, (\pi_i, q_i)$

A_i is the LHS symbol of CF production π_i.

In addition, an existing string γ will, in some cases, be extended by concatenating it with some particular α_i. The resulting string, $\gamma\,\alpha_i$ will be referred to as β_i. It should be noted that α_i and β_i may, in general, be empty strings. The maximum length of the strings to be generated is assumed to be k.

A single FPL production corresponds exactly to a set of RHS occurrences of a specific symbol in the set of CF productions. This set consists of only one RHS occurrence if the FPL production is not the result of combination. What must be done, then, is to show that the lookahead construction accurately generates those terminal strings which may follow the appropriate RHS occurrences.

The first step in generating lookahead is to initialize an appropriate number of strings. Assuming the set of descriptors for the FPL production P in question is

$$S = \{ (\pi_i, n_i) \mid i = 1,2,\ldots, m \}$$

the strings which must obviously follow immediately are precisely α_i for $i = 1,2,\ldots, m$. A review of the buffer initialization as described in chapter 5 shows that these are precisely the strings initially generated.

Two other steps must be taken. The first is to extend those strings which are not of sufficient length. This is done by the procedure GETSYM. The second is to change the mixed strings (i.e., containing both terminal and nonterminal symbols) into purely terminal strings. This is done by the procedure NT2T. The sequence in which

these two procedures are applied, the comparison of generated terminal strings with those strings which might follow precluded FPL productions and the methods which minimize string length can vary greatly depending on implementation and will not be dwelt upon here. Those readers interested in understanding and verifying the specific implementation used are invited to study the flow charts of Figures 1 and 2 in chapter 5.

Of the two steps yet to be described, the conversion to terminal strings is the simpler and will be discussed first. A specific occurrence of a nonterminal symbol in a lookahead string may be eliminated by generating a new string for each definition of that nonterminal with the occurrence replaced by its definition. The original string is then eliminated from the set of lookahead strings. This process must be repeated for all remaining nonterminal symbols in the lookahead strings and for any new nonterminals introduced by the replacement.

Unfortunately, left recursion in the CF grammar could cause this iterative replacement to be a never ending process. If, however, each new string thus created, is compared to all other lookahead strings and, more importantly, to those strings already processed and eliminated (but saved for this purpose) and ignored when found to be identical to one of these previously existing strings, the process must end. This is because the finiteness of k and of the alphabet ($V = V_N \cup V_T$) guarantees that there can be only a finite number of strings.

The method just described guarantees the accurate conversion of mixed strings to terminal strings. A review of the procedure NT2T and its use as described in chapter 5 shows that this is precisely the method used in the lookahead construction of that chapter.

The most important step in generating the proper set of terminal symbol lookahead strings is the proper extension of those initial (or partially extended) strings which are insufficient in length. In general, it is more efficient to do all possible nonterminal symbol elimination prior to any string extension since this process can and usually does cause some string extension.

In extending strings it is important to recognize that each initial string which requires extension is the completion of some RHS. In particular, it comes from the CF production numbered π_i. Thus what can follow at this point is exactly what can follow A_i, the nonterminal which is defined by production π_i. Assuming that A_i has RHS occurrences at (π_j, n_j) for $j = 1,2,\ldots, p$, the single string α_i must be replaced by the set of strings $\alpha_i \alpha_j = \beta_j$. This process must be repeated for each initial α_i. Each lookahead string is now β_j for some j.

Some of these β_j will still be of insufficient length. In fact, in those cases where $n_j = q_j$, the initial $\alpha_i = \beta_j$ and the string has not been extended at all. Each β_j is the end of the RHS of the production numbered π_j. It can be extended by adding all those strings which follow RHS occurrences of the nonterminal symbol A_j in the same manner as was described above for adding those symbols which followed RHS occurrences of A_i. This process is then repeated until all strings are of sufficient length.

This process must eventually end for the same reason that the nonterminal to terminal conversion process must end. It must be noted, however, that two identical strings, β_{j_1} and β_{j_2}, are not considered the same unless $j_1 = j_2$. This is true for both the conversion and the extension processes.

The method described above accurately extends lookahead strings which are of insufficient length. The procedure GETSYM of chapter 5 does not, however, operate in precisely this manner. It has two differences which increase the efficiency of extension without affecting the accuracy.

The first of these differences minimizes the importance of j for each β_j. It must be remembered that each j corresponds to a particular (π_j, n_j) which, in turn, corresponds to a particular A_j. A review of the procedure will show that once a string becomes β_j through addition of α_j, the only importance of j is that it determines A_j. Thus the procedure GETSYM keeps track of neither j nor (π_j, n_j), but rather associates with each string, the appropriate A_j. This necessitates less saving of information since the descriptors (π_j, n_j) are no longer necessary. It also increases the number of cases where newly created strings can be ignored since two identical strings β_{j_1} and β_{j_2} may be considered the same even if $j_1 \neq j_2$ when $A_{j_1} = A_{j_2}$.

The second thing which GETSYM does differently is to use tail symbol relationships to eliminate those iterations of the extension procedure which fail to extend the string (i.e., when $n_j = q_j$ for some j). Assume that the following is a subset of the CF productions:

$$
\begin{array}{lll}
A_1 & \rightarrow & \ldots A_2 \\
A_2 & \rightarrow & \ldots A_3 \\
 & \vdots & \\
A_{n-1} & \rightarrow & \ldots A_n
\end{array}
$$

and A_n is the nonterminal whose definition (RHS) is ended by the string to be extended. With the method described above, at least n iterations would be required since one of the new strings generated by each attempt would not extend the string but simply change the nonterminal symbol associated with it (i.e., A_i to A_{i-1} for $i = 2, \ldots, n$). Clearly the ultimate result is to extend the string with those symbols which follow A_i for $i = 1, \ldots, n$. It is also clear that A_n is a tail symbol of each A_i for $i = 1, \ldots, n$ (see chapter 2). GETSYM makes use of this fact by extending the string on the first iteration with those symbols which follow any occurrence of any nonterminal symbol B which has A_n as a tail symbol (in this case all A_i for $i = 1, \ldots, n$) as opposed to any occurrence of A_n only. This allows GETSYM to ignore occurrences of B which end a RHS (i.e., do not in fact extend the string). This obviously accomplishes the same thing as the method originally described but in a more efficient manner.

The only parsing decision which must be made when a set of FPL productions is used as a parsing algorithm is to decide which production applies. It has now been shown that when this decision can be made by lookahead, it is correctly made by the set of FPL productions generated by the CF to FPL conversion algorithm of chapters 3 through 5.

6.3 Implicit Lookback Adequacy

It remains to be shown that all parsing decisions based on that part of the input already partially parsed will be correctly made by a set of FPL productions as generated by this conversion algorithm.

This is a somewhat more complex situation since the lookback is not explicit, but is implicit in the grouping of the FPL productions. The approach to be used will be to show that all groups contain all and only those productions which could possibly apply upon transfer to the group. When this is complete, the adequacy of the lookback will have been proved because the exact accuracy of the groups implies that there is at least one particular partially parsed string which could precede the application of any and all the productions in the group.

The two special groups, START and NT (0,0), are simple mechanical devices for beginning and ending the parse. A review of the contents of these groups (section 3.3) will verify that they contain all and only the appropriate productions. The remainder of the groups fall into eight distinct types (T, CT, TH, CTH, NTH, CNTH, NT and CNT) and the proof, therefore, has the following eight distinct cases:

case 1: T (π, n) groups

The FPL production which generates the transfer to the T (π, n) group has descriptor $(\pi, n-1)$ (3.3.2) and applies only when symbol n-1 on the RHS of the CF production numbered π has just been recognized. The transfer is to T (π, n) if, and only if, symbol n of CF production π is a terminal symbol. Thus the T (π, n) group must attempt to recognize exactly and only that terminal symbol. The descriptor set $D_{T(\pi, n)}$ is, by definition, the single descriptor (π, n) (3.2.4) which maps into the FPL production which tests for the appropriate terminal symbol and then takes action based on the symbol which may or may not appear at position n + 1 of CF production π. Thus, the

T (π, n) group contains exactly the single FPL production which could apply upon transfer to the group.

case 2: CT (m) groups

The FPL production which generates the transfer to the CT (m) group is a combination of FPL productions of the form which generate transfers to T (π, n) groups (section 4.1). Thus, the CT (m) group must contain exactly those productions which would have made up the appropriate T (π, n) groups. The integer m has associated with it precisely this set of labels, L_m (4.1.1). The descriptor set $D_{CT(m)}$ is defined (4.1.2) as the union of the descriptor sets of these appropriate T (π, n) groups. Thus, the CT (m) group contains almost by definition, exactly those productions which could apply upon transfer to the group.

case 3: TH-A groups

Transfer is made to a TH-A group of FPL productions upon recognition of the fact that some derivative of nonterminal A must follow if the input is correct (3.3.1 and section 4.2). Since the symbols which come next are from the input stream, they must be terminal symbols. The FPL productions of the TH-A group must test for all and only those occurrences of terminal symbols which may begin some derivative of A.

The definition of descriptor set D_{TH-A} (3.2.1) explicitly specifies that all the elements of D_{TH-A} must refer only to terminal symbols which are first symbols on RHS's of CF productions. Thus, the FPL productions of TH-A cannot be testing for nonterminal symbols

or occurrences of terminal symbols which do not begin derivatives. For an occurrence of terminal symbol t to begin a derivative of A_1, the following relationship must exist in the set of CF productions:

$$\begin{array}{lcl} A_1 & \rightarrow & A_2 \ldots \\ A_2 & \rightarrow & A_3 \ldots \\ & \vdots & \\ A_{n-1} & \rightarrow & A_n \ldots \\ A_n & \rightarrow & t \ \ldots \end{array}$$

A comparison of this relationship with the definitions of D_{TH-A} and of head symbols (chapter 2) makes it obvious that the descriptors of D_{TH-A} refer to all and only those occurrence of terminal symbols which may begin derivatives of A. The descriptor to FPL production mapping rules (section 3.3) then guarantee that group TH-A contains all and only the appropriate FPL productions.

case 4: CTH (m) groups

As was the case in the CT (π, n) type groups, the transfer to a CTH (m) group is made by a FPL production which is the result of combining two or more FPL productions of a single type, namely, those which would initiate recognition of nonterminal symbols by transfer to TH-A groups. Thus, a CTH (m) group must contain exactly those productions which would be in the various TH-A groups.

Each uncombined FPL production which causes transfer to a TH-A group also creates and pushes into the marker stack the symbol

$\bar{A}$ (π, n) or $\bar{A}$* (m). Thus, a set of such productions must create a set of such marker symbols, $\bar{A}_1$ ( ), $\bar{A}_2$ ( ),..., $\bar{A}_n$ ( ) and cause transfer to TH-A_1, TH-A_2, ..., TH-A_n. The symbol $(\bar{m})$ is created by a combination of productions of this type and is defined to be this set of simple marker symbols (section 4.2). Thus, if $\bar{A}_i$ (π, n) $\in$ $(\bar{m})$, then all the productions of TH-A_i must be in the CTH (m) and no productions other than those which are in some appropriate TH-A_i group should be in CTH (m). It is clear from the definition of the descriptor set D_{CTH} (m) (4.2.4) that this is the case.

case 5: NTH-A groups

An NTH-A group is reached by a dynamic transfer DNT-B when there has just been a reduction to the nonterminal B and the top symbol on the marker stack is $\bar{A}$ (π, n) or $\bar{A}$* (m). The marker for A was pushed when the search for an A began. If the A has been completely recognized, processing of the appropriate CF production RHS has continued. If there are other nonterminal symbols following the A, then the corresponding markers would be pushed onto the marker stack. When the end of the CF production has been recognized, then all marker symbols for nonterminal symbols on its RHS are popped from the marker stack. Thus, the existence of a marker symbol for A at the top of the marker stack means that the B just recognized must be part of some derivation of A.

The B must be the first symbol of the RHS of some CF production or else the beginning of the search for it would have caused a marker for it to be pushed on top of the A marker. The LHS of the CF production, C, which begins with B must be a head symbol of A. That this must be true is seen by the definition of head symbol and the fact

that B must be part of some derivative of A. If this were not true, the following relationship would have to exist:

$$
\begin{array}{lcl}
A & \rightarrow & A_1 \ldots \\
A_1 & \rightarrow & A_2 \ldots \\
 & \vdots & \\
A_{n-1} & \rightarrow & \ldots A_n \ldots \\
A_n & \rightarrow & A_{n+1} \ldots \\
 & & \vdots \\
A_{n+m} & \rightarrow & C \ldots \\
C & \rightarrow & B \ldots
\end{array}
$$

in which case a marker for A_n would be pushed onto the marker stack above the A marker which is, in fact, found at the top.

Thus, when a nonterminal symbol has just been recognized which does not match the simple or special marker which is at the top of the marker stack, it must be the first RHS symbol of a CF production whose LHS is a head symbol of the nonterminal whose marker is atop the marker stack. By definition, this is precisely the set of symbols referred to by the descriptors of NTH-A (3.2.2).

case 6: CNTH (m) groups

The transfer to a CNTH (m) group is made when a nonterminal B has just been recognized and put in the parser symbol and the set $(\overline{m})$ of marker symbols does not contain $\overline{B}$ (π, n) or $\overline{B}*$ (n). Using the same arguments as those of case 5, it is clear that the B just recognized must be the first RHS symbol in a CF production whose LHS is a head symbol of a nonterminal symbol, the search for which was initiated when $(\overline{m})$ was pushed onto the marker stack.

The $(\overline{m})$ represents a set of simple and special marker symbols corresponding to set of nonterminal symbols, $A_1, A_2, \ldots, A_n$, (section 4.2) which could come next at the time the $(\overline{m})$ was pushed onto the stack. Thus, the set of FPL productions which could possibly apply is precisely the union of the FPL productions of the various NTH-A_i. This is equivalent to saying that the descriptor set $D_{CNTH(m)}$ must equal the union of descriptor sets $D_{NTH\text{-}A_i}$ for all A_i such that $\overline{A}_i$ $(\pi, n) \in (\overline{m})$ or $\overline{A}*$ $(n) \in (\overline{m})$ and this is precisely the definition of $D_{CNTH(m)}$ (4.2.5). Thus, CNTH (m) contains all and only those FPL productions which could apply on transfer to the group.

case 7: NT (π, n) groups

Transfer is made to an NT (π, n) group when a nonterminal symbol A has just been recognized and the top symbol on the marker stack is $\overline{A}$ (π, n) or $(\overline{m})$ where $\overline{A}$ $(\pi, n) \in (\overline{m})$. If $(\overline{m})$ is at the top and $\overline{A}$ $(\pi, n) \in (\overline{m})$, we must be at some stage in recognizing the A which is the n^{th} symbol on the RHS of CF production π. It cannot be part of the derivative of some other nonterminal symbol B where $\overline{B}$ $(\pi_1, n_1) \in (\overline{m})$ or $\overline{B}*$ $(m_1) \in (\overline{m})$. This is due to the insertion of dummy symbols as

described in section 4.3. As seen in case 5, if the A just recognized were part of a derivation of B, it would have to be a head symbol of B. The case where A is a head symbol of B and the two FPL productions with descriptors $(\pi, n-1)$ and (π_1, n_1-1) could be combined is precisely the case where A and all symbols following it in CF production π are replaced by a dummy nonterminal symbol D and $\overline{A}\ (\pi,n)$ would not be an element of $(\overline{m})$. Thus, in either the $\overline{A}\ (\pi,n)$ or $(\overline{m})$ case, the A just found must be either the A corresponding to (π, n) or some part of a derivative of that particular A.

If the A just found is not the one corresponding to the n^{th} symbol of CF production π, then it must by the reasoning of case 5 be a beginning of some derivative of A (i.e., a head symbol of A). Thus, by definition it must be a left recursive occurrence of A. The NT (π, n) group must, therefore, contain exactly the (π, n) occurrence of A and all left recursive occurrences of A. The descriptor set $D_{NT(\pi,\ n)}$ contains by definition (3.2.3) precisely the descriptors for these occurrences.

case 8: CNT (m) groups

A CNT (m) group is transferred to, when a nonterminal symbol A has just been recognized and the top symbol on the marker stack is $\overline{A}^*\ (m)$ or $(\overline{k})$ where $\overline{A}^*\ (m) \in (\overline{k})$. All of the arguments of case 7 apply again with the single (π, n) occurrence of A replaced by a set $S = \{(\pi_i, n_i) \mid \overline{A}\ (\pi_i, n_i) \in \overline{A}^*\ (m)\}$ of CF occurrences of A.

The descriptor set $D_{CNT(m)}$ must contain exactly all $(\pi_i, n_i) \in S$ and all descriptors for left recursive occurrences of A.

This is precisely the union of all $D_{NT(\pi_i, n_i)}$ $((\pi_i, n_i) \in S)$. But this in turn is exactly the definition of $D_{CNT(m)}$ (4.2.2).

6.4 Conclusion

It has been shown that each group of FPL productions contains exactly those productions which could possibly apply based on what has preceded the transfer to the group. Therefore, no parsing decision can be directly affected by further investigation of the partially parsed preceding string. It is also clear that all possible use has been made of the next k input symbols through the explicit search (when necessary) for all k symbol strings which could follow if a particular FPL production could be applied.

7. RELATIVE EFFECTIVENESS

The second step in evaluating the effectiveness of the conversion algorithm is to compare its results with other recently developed methods for generating parsing algorithms from context free grammars. This comparison will be done on three levels. First will be the development of an expression for the upper bound, U, on the parsing time required by each of the parsing algorithms generated. Second will be an evaluation of the effect, on these expressions of replacing the upper limit variables with variables representing averages. The third approach is a subjective analysis which attempts to point up the strengths and weaknesses of each algorithm relative to the others.

Two of the methods to be compared are those of DeRemer [15] and Korenjak [16] because they are relatively recent and the parsing algorithms which result from these methods operate by applying the same basic operations as the result of CF to FPL conversion. The classic LR (k) recognition algorithm of Knuth [14] will also be compared since it is the most widely known one which also uses these same basic operations. Lastly, the method of Earley [17] will be compared since it is the most widely known among recent attempts at automatic recognition generators.

7.1 Upper Bounds

The algorithms of DeRemer, Korenjak and Knuth and the CF to FPL conversion result all operate by applying various sequences of what will be referred to as basic operations. These operations are:

1. test for a given symbol
2. put a symbol into stack or word
3. increment stack pointer
4. decrement stack pointer (remove symbols from stack)
5. scan symbol from input
6. transfer control

These operations correspond roughly to basic machine operations and will thus be treated as being of equal complexity.

The variables used in the upper bound expressions are defined as follows for an arbitrary grammar G and input stream S:

t = number of different terminal symbols in G

n = number of different nonterminal symbols in G

p = number of productions in G

l = number of symbols on the RHS of the longest production in G

k = lookahead length used

m = number of symbols in S

r = number of reductions in the longest possible sequence of reductions without an intervening scan from S (a function of G and S)

q = maximum number of occurrences of a single nonterminal in the RHS's of productions of G.

The upper bound expressions for all except the algorithm of Earley are calculated by assuming that for each symbol scanned from S, the longest possible sequence of reductions must be made. It is also assumed that when comparisons of any sort are necessary, the maximum number of alternatives are tested sequentially with the last possibility being correctly matched.

7.1.1 FPL Production Set

This section shows in some detail the calculation of the upper bound, U_c, on the number of basic operations necessary to parse input S with the parsing algorithm generated by the CF to FPL conversion applied to grammar G.

For each symbol of S there must be a scan from the input stream, a placement of the symbol into the parser word, a transfer to a TH, CTH, T or CT group, and a sequence of reductions. The upper bound is then

$$U_c = m\ (3 + Y + (r-2)\ X + Z)$$

where Y represents the number of operations associated with the first reduction, Z the number of operations associated with the last reduction and X the maximum number of operations associated with each of the intervening reductions.

The first set of operations contributing to Y is a test of the parser word for the appropriate terminal symbol of which there are t possibilities. Then all possible lookahead strings must be compared symbol by symbol which involves kt^k comparisons. This is followed by

placing the new nonterminal into the parser word, popping the marker stack, comparing it to the top marker word and transferring to the appropriate NTH group (always the worst case). Thus, the maximum is

$$Y = t + kt^k + 4$$

In the NTH group all but one of the nonterminals is a possibility (the symbol on the top of the marker stack is not), necessitating n - 1 comparisons. There are kt^k lookahead operations as before. The marker stack is not popped in this case since the reduction being made is of only one symbol (otherwise we would be in a NT as opposed to NTH group), but the other three concluding operations apply. Thus, the maximum is

$$X = n - 1 + kt^k + 3$$

The calculation of Z begins with the same parser word and lookahead operations. This may be followed by adding to the marker stack and incrementing its pointer at which time the next scan would follow. Thus, the maximum is

$$Z = n - 1 + kt^k + 2$$

Substituting for X, Y and Z and simplifying gives

$$U_c = m\ (rkt^k + rn + 2r + t - n + 4)$$

which represents the maximum number of basic operations necessary for the result of the CF to FPL conversion of G to parse S.

7.1.2 Comparison

The parsing algorithms generated by Korenjak have the same upperbounds as those generated by Knuth since Kornejak differs from Knuth only in ways which do not affect the number of operations necessary for parsing. The upper bound for DeRemer, U_D, and the upper bound for Knuth and Korenjak, U_K, were calculated in the same manner as U_c was calculated. That is, a thorough step by step analysis of the maximum number of basic operations necessary between scans.

The upper bound on algorithms generated by Earley, U_E, is not calculated in this manner since they do not operate by executing the same basic operations. That upper bound is taken directly from the work done by Earley himself. U_E does not represent basic operations and this author believes that it would be necessary to multiply the expression by a constant factor of at least five to make it comparable. The other three upper bounds, U_c, U_D and U_K, are, however, attempts to at least approach the least upper bound in each case. This is not true of U_E which is calculated by Earley only to show a proportionality to m, the number of symbols in the input stream S. The assumption is then, that these two factors tend to cancel and that U_E is roughly comparable to the other three.

The upper bounds are:

$$U_c = m\ (rkt^k + 2r + t + 4) + m\ (rn - n)$$

$$U_D = m\ (rkt^k + 2r + t + 4) + m\ (kt^k + rq + 4r + 2)$$

$$U_K = m\ (rkt^k + 2r + t + 4) + m\ (kt^k + rn + rt + 3r + n + t + 2)$$

$$U_E = m\ p^3\ l^2\ t^{2k}$$

The major differences in these expressions will be explained in section 7.3, during the subjective analysis of the methods and their properties.

7.2 Average Times

The average values of most of the variables involved are, in fact, almost impossible to ascertain since they depend to such a great degree on specific grammars and input streams. This section will attempt to show the effect on U_c, U_D and U_K of considering the realistic cases (with estimates based on experience). The upper bound of Earley, U_E, will not be considered here since it only vaguely represents the same thing as the other three and because it is calculated in a completely different manner.

The first significant change is brought about by recognizing that each scan of a new symbol from the input stream does not result in r consecutive reductions. It is estimated rather that there are approximately two reductions for each input symbol.

The next change stems from recognition of the fact that when a terminal (or nonterminal) is possible, it is not necessary to assume that the last of all t (n) possibilities will apply. It is estimated from experience that the average number of possibilities in such cases is no more than three or four. The average number of tests necessary is approximately two.

In the CF to FPL conversion method and the method of DeRemer, lookahead is used only when necessary and then only as far as necessary. Rarely is a lookahead length of more than one required. In fact, zero is the most common. Therefore, kt^k will be replaced by two in U_c and U_D. The methods of Knuth and Korenjak require the use of lookahead to its maximum length in every case. The value of kt^k is, therefore, extremely dependent on the specific grammar and could, in fact, reach as much as one thousand or more. For this comparison we will be more conservative and use 50 as the number of symbols which must be compared when lookahead is applied.

The final change stems from recognition that not all nonterminal symbols appear in the grammar q times as RHS symbols. It is assumed that the average number of such appearances is four, of which only half must be tested.

The upper bound expressions U_c, U_D and U_K become approximate average time expressions T_c, T_D and T_K by making the following substitutions:

kt^k replaced by 2 in U_c and U_D,

kt^k replaced by 50 in U_K and

r, t, q and n replaced by 2.

The time expressions calculated in this manner are

$$T_c = 16\ m$$

$$T_D = 30\ m$$

$$T_K = 180\ m$$

7.3 Subjective Evaluation

The CF to FPL conversion algorithm described here and the systems of DeRemer, Knuth, Korenjak and Earley use quite different notations and terminology. The discussion of these systems which follows will use, as much as possible, the terminology developed to this point for the conversion method when referring to roughly equivalent activities of the resulting parsing algorithms.

Again the methods of Earley must be considered independently of the others. The main reason for this is that the algorithm which he describes, and from which his upper bound is calculated, generates recognizers as opposed to parsers. That is to say, his method determines whether or not a given string is in a particular language without necessarily determining the specific sequence of reductions which would convert the string to an objective symbol. He states that the algorithm can be converted to a form which generates parsers, but it is not clear exactly what form these parsers would take. It is thus not possible to compare them to parsers generated by the other methods.

A second reason that comparison of Earley's method to the others would be unreasonable is that his goal appears to be proof of the existence of an algorithm which generates recognizers with certain properties. This is as opposed to the others (with the possible exception of Knuth) who are attempting to create practical parsing algorithm generators. The basic approach is to construct, at recognition time, the equivalent of the group of FPL productions (without transfers) which could apply next. This construction is based on the immediately preceding group, other preceding groups which are identified by use of a

mechanism similar to the marker stack and the CF grammar. It seems to this author that a parsing algorithm based on similar methods would be relatively slow and inefficient.

The other four methods are similar in many respects. They do, however, have differences which are significant enough to account for the variations in the time expressions of section 7.2. The most important of these is the unnecessary use of lookahead which has already been described.

Another significant difference is the manner in which Knuth (and therefore, Korenjak) separates the parsing decisions from the decisions involved in transferring to the next step. The basic parsing decision is to reduce n symbols at the top of a stack to a single nonterminal symbol or to scan another terminal symbol from the input stream into the top of the stack. This decision is based solely on k symbols of lookahead. The transfer decision, however, is based solely on the symbol at the top of the stack (in this case a single stack holds both markers and parsing symbols). This leads to a significant amount of unnecessary testing. The most obvious example of this is when a one symbol lookahead recognizes the specific symbol t which causes t to be scanned into the stack. The next step is to test the symbol at the top of the stack against many possibilities (one of which is t) to determine the next transfer address.

A basic difference between the method of Knuth and the CF to FPL conversion method was described in chapter 6 as the reason that the conversion method method cannot process all LR (k) grammars. Korenjak's approach is to take an arbitrarily selected subset of the nonterminals (presumably those which occur most frequently in the grammar) and use a single parsing procedure for each of these. When successful in the

selection process, the result is a parsing speed the same as Knuth, but a smaller set of tables. The conversion method and DeRemer's method attempt to do this with all nonterminals which, when successful, minimizes, to a great degree, the table sizes. As noted in chapter 6, this does, unfortunately, have the added effect of restricting the class of grammars which can be successfully processed. DeRemer does introduce a technique in his method which duplicates the recognition of specific occurrences of nonterminals when required, but he admits that implementation is completely impractical. Unsuccessful attempts have been made to accomplish this in an implementable way in the CF to FPL conversion algorithm.

The difference between T_C and T_D is explained by three factors. The first is in the dynamic transfer. The DeRemer method tests the marker stack to determine which, of all possible uses of the nonterminal just recognized is, in fact, the one which applies. It is believed that the conversion method is superior on the average because the number of possibilities in an NTH group is no greater than the number of occurrences of a nonterminal and the marker stack is used in such a manner as to force testing of the most likely possibility first.

The second increase in the number of operations required by a DeRemer generated parser is caused by the fact that a marker symbol is pushed for every new group which is entered. This means more pushes and also that a pop accompanies every reduction. The conversion algorithm, on the other hand, pushes marker symbols only for transfers to TH and CTH groups and has some reductions which require no pop of the marker stack. This relative weakness of the DeRemer method also applies to the Knuth and Korenjak methods.

The third factor is that a reduction is separated from the following dynamic transfer in the DeRemer method. This requires the execution of an apparently unnecessary transfer. There are several other such relatively minor weaknesses in the methods of DeRemer, Knuth and Korenjak which presumably could and would be eliminated in any meaningful attempt at implementation.

7.4 Summary

This chapter has attempted to show the differences between the parsing algorithms generated by the CF to FPL conversion method described in chapters 3 through 5 and the parsing algorithms generated from CF grammars by the methods of Earley, Knuth, Korenjak and DeRemer.

The method of Earley must, in fact, be considered separately from the others since it generates recognizers as opposed to parsers. In addition, it appears to be a proof of the existence of a method for generating recognizers with certain theoretical properties as opposed to a method which might be implemented for practical purposes.

The method of Knuth is also an existence proof. It, however, would be of practical value if it could be implemented efficiently enough. The basic problem is simply the size and quantity of the tables generated. The method of Korenjak is similar to that of Knuth except that it treats an arbitrarily selected subset of the nonterminal symbols of the CF grammar as special cases. If the appropriate nonterminals are selected, this method will generate parsing algorithms for all LR (k) grammars as accepted by Knuth.

The conversion method and the DeRemer method eliminate this trial and error nonterminal selection by, in effect, assuming that all the nonterminal symbols fall in this special class. This, unfortunately, has the effect of reducing the class of grammars for which parsing algorithms may be generated (see chapter 6).

Section 7.1 calculates expressions for the upper bounds on the parse times of algorithms generated by the marious methods. These expressions were arrived at in such a manner as to approach as nearly as possible, the least upper bounds. Section 7.2 calculates expressions for the average parse times of the algorithms generated. In each case the parsing algorithms generated by CF to FPL conversion appeared most efficient.

8. THE IMPLEMENTED TWS

This chapter will be a brief and incomplete description of the translator writing system (TWS) which has been implemented using the CF to FPL conversion algorithm as the basis of the syntax preprocessing phase. The emphasis is on those factors which directly affect or are directly affected by the conversion. This chapter is included so the reader can evaluate the algorithm in a practical sense and also as a first step for potential users of either the conversion algorithm or the implemented TWS.

The TWS has been implemented on a Burroughs' B5500 and is written entirely in Burroughs' version of ALGØL for this machine. Certain of the implementation methods were developed specifically to make use of particular features of this machine and language.

8.1 Syntax

The syntax preprocessing part of the TWS consists of three phases. The first phase is a conversion of the input grammar, written in a metalanguage called TWINKLE [20], to an internally represented set of CF productions. This metalanguage uses notation very similar to Backus-Naur Form (BNF) but extends it to include productions whose RHS's resemble regular expressions. This allows grouping of alternatives within a single RHS and the specification of arbitrarily long lists of symbols without the use of recursive productions. TWINKLE also allows the replacement of most of the special metasymbols with English language special words. For example, the set of CF productions

<E> ::= <E> + <T> ;

<E> ::= <E> - <T> ;

<E> ::= <T> ;

is equivalent to the TWINKLE production

AN <E> IS DEFINED AS A LIST ØF
<T>S SEPARATED BY [+ ØR -] ;

It is felt that use of English forms of TWINKLE will simplify the language designer's task of documentation.

Appendix A is a listing of the English-like TWINKLE specification of the syntax of a small subset of ALGØL called DEMALGL (DEMonstration ALGØL). This is converted by the TWS built TWINKLE compiler into the set of CF productions of Appendix B. The translation from TWINKLE to CF productions also includes the following:

1. Elimination of all empty RHS's.
2. Elimination of all nonterminals with only a single definition.
3. Rearrangement of the CF production table to a group the definitions of each nonterminal symbol.
4. Calculation of the head symbol tables.
5. Insertion of dummy nonterminal symbols (as described in section 4.3).

6. Creation of a table of LHS and RHS occurrences of all nonterminal symbols.

7. Insertion of the special production defining Σ.

The central processor time for the TWINKLE compilation of DEMALGL on the Burrough's B5500 was 64 seconds.

Appendix B is PRØTAB (CF PRØduction TABle) as output from the TWINKLE compiler. PRØTAB is stored internally as a one-dimensional array with a single word corresponding to each RHS symbol or semantic call. This word contains the following bit fields:

one bit specifying whether or not this is a left recursive occurrence of a nonterminal.

one bit specifying whether or not this is a nonterminal which is the last nonsemantic RHS symbol.

one bit specifying whether or not the next symbol is a semantic call.

twelve bits for the number of the nonterminal LHS of this production.

six bits for the type of this symbol (terminal, nonterminal, semantic test, etc.)

twelve bits for the number of this symbol (symbols are numbered sequentially within each type).

The PRØTAB of Appendix B differs from the CF production table described in chapters 3 through 5 in that the descriptors are

not of the (π, n) form but are single numbers corresponding to addresses in PRØTAB. The (π, n) form was used to make the description of the algorithm more easily understood.

The dummy nonterminal defined at location 310 is an example of the process described in section 4.3. The potential bar to combination was the head symbol relationship between <arithmetic expression> now at location 310 but originally at location 243 and the occurrence of <boolean expression> at location 276.

The second phase of the syntax preprocessing is the conversion of the internally represented set of CF productions to an internally represented set of FPL productions, using the algorithm described in chapters 3 through 5. As noted in Chapter 4, it is an implementation feature which dictated the use of one symbol lookahead as the primary method of eliminating preclusion. This feature is the use of bit patterns to represent sets of terminal symbols. These bit patterns are pointed to by a third type of symbol called an "any" symbol. These sets can be specified directly in the TWINKLE metalanguage. They are also constructed internally to combine a set of FPL productions within a group which are identical except that the parser symbols are different terminal symbols. Their most important use is an internal combination of a set of one symbol lookaheads into a single "any" symbol which requires only a single test at parse time. The table of terminal head symbols created during the TWINKLE conversion is in the same bit pattern form and allows a simplification of the routine NT2T (chapter 5) when a nonterminal is being expanded in the k-th position during k symbol lookahead construction.

Appendix C is the set of FPL productions which results from a conversion of the CF productions of Appendix B. The reference to "top stack symbol" is synonymous with the term parser symbol as used throughout this work. The phrase "is input" is followed by the set of lookahead strings associated with a production. A "bar" is a simple marker symbol, "sbr" is a special marker symbol and "cbr" is a combined marker symbol. The descriptor associated with a simple marker symbol, the number associated with a special marker symbol and the number of marker symbols to be popped at a reduction are not included in this listing.

Points worthy of note in this example include the various types of combination. The combined groups begin at FPL production 325 and each identifies the group which contains the combined production.

Also important in terms of the efficiency of the parse is the insertion of TPN groups (which by definition contain only one production) inline as opposed to being separate groups requiring a transfer. An example of this occurs in productions 44-46 in the TH-6 group. In this case, if the parser symbol is, in fact, <*I>, then production 44 applies but transfers to production 45 (instead of a separate TPN-191 group) upon completion. If production 45 does not apply, control is transferred to the error production at the end of the group. If the parser symbol is not initially <*I>, then production 44 does not apply, but control skips TPN production 45 and tries to apply production 46 next. This becomes important when it is realized that large languages require paging of a segmented pseudo instruction table. Thus I/Ø is minimized when transfers are relatively local. Another advantage is that the parser symbol need not be checked in the TPN production if the preceding production uses lookahead. This fact would be almost impossible to ascertain if the

TPN production was a separate group which could be transferred to from more than one place.

The central processor time for this phase of syntax preprocessing was 76 seconds.

The final phase of syntax preprocessing is a conversion of the internally represented set of FPL productions to a sequence of pseudo orders which may be used directly for interpretive parsing or may be further converted into compilable Burrough's B5500 ALGØL code. The choice between interpretive parsing and parsing with executable code is made by the language designer. Which choice is most efficient is a function of the grammar (at least with the particular implementation now in use). This phase is the most machine dependent since it makes maximum use of the features of the B5500. It is also in this phase that a number of other effective optimizations are made to the set of FPL productions. These optimizations are significant enough to lower the parse time by 25 to 30 per cent. The DEMALGL example of Appendices A through C was converted only to pseudo instructions. The time for this phase of syntax preprocessing was 23 seconds.

8.2 Semantics

The TWS works by calling a user specified semantic routine at the appropriate point in the parse. These calls may be placed at any point in a TWINKLE production except at the beginning. They are then associated with the preceding symbol in the resulting set of CF productions. When that particular use of the symbol has been recognized, the semantic routine is called into execution. This has a significant effect on the CF to FPL conversion algorithm when the semantic call follows a

symbol other than the last one on an RHS. Under these circumstances the FPL production whose corresponding descriptor points to that symbol cannot be combined with any other FPL production which does not have the same semantic call. It is relatively obvious that the parsing decision must be made at this point, and in this sense it is equivalent to the parsing decision necessary upon completion of a RHS. Intuitively, in fact

$$A \rightarrow a \text{ @SX } B$$

is, from the users point of view, equivalent in this sense to

$$A \rightarrow D \; B \text{ and}$$
$$D \rightarrow a \text{ @SX}$$

where @SX is the specification of a call on semantic routine X.

A second use of semantics has potentially an even greater effect on the CF to FPL conversion and resultant parser. This is a semantic test or condition and is specified @TX. A semantic test routine may do anything that a normal semantic routine may do, but it must set a global boolean variable to true or false. This type of routine is invoked after recognition of the appropriate symbol, but before it has been ascertained what specific use of the symbol applies. The use of this type routine is, in fact, to differentiate in a case where no syntactic differentiation is possible using the CF to FPL conversion method. The semantically conditioned terminal (or nonterminal) symbol is, for conversion purposes, considered to be a different symbol from

the same terminal (or nonterminal) symbol with no semantic test or with a different semantic test. The following is an example which might appear in the grammar for an ALGØL type language in the specification of boolean and arithmetic assignment statements:

1. <AS> ::= <BAS> ;
2. <AS> ::= <AAS> ;
3. <BAS> ::= <BLHS> = <BE> ;
4. <AAS> ::= <ALHS> = <AE> ;
5. <BLHS> ::= <*I> ;
6. <ALHS> ::= <*I> ;

where <*I> represents the terminal symbol "identifier" (recognized by a scanner). The TH-<AS> group would include

<*I> | → <BLHS> (0) | DNT-<BLHS> and

<*I> | → <ALHS> (0) | DNT-<BLHS>.

This is an unresolvable preclusion. CF production five could be conditioned

<BLHS> ::= <*I> @ TX

where semantic test routine X checks some previously built symbol table and returns a value of true if and only if the particular identifier in question has been declared BØØLEAN. In this case, the resulting FPL productions

<*I> @ Tx | ... and

<*I> | ...

are considered to have two different parser symbols and are, therefore, not in conflict.

The use of this semantic feature adds tremendously to the parsing power of the total system since semantic routines (both test and normal) have access to all parts of the total system. It is possible, for example, to go to the following extreme in building a compiler for ALGØL. Let

<PRØGRAM> ::= <u>BEGIN</u> @ T 1 ;

be the entire grammar and @ T1 be a handwritten ALGØL compiler which uses the TWS supplied scanner and returns true or false depending on whether or not the input stream is an acceptable ALGØL program. Clearly the language designer must take care in using this feature, since the existence of a semantic test causes the conversion algorithm to assume complete differentiation between the conditioned occurrence of a symbol and all other occurrences of the symbol which are not similarly conditioned.

The semantic routines are written in Illinois Semantics Language (ISL) [21] which includes all of Burrough's B5500 ALGØL as a subset. In addition, there are a number of special constructs which simplify the declaration and use of a number of data structures such as stacks and tables which are frequently used in constructing compilers.

There are also special constructs which allow the semantics writer to output an intermediate language stream of operators (next pass semantic routine calls) and operands.

The semantic preprocessor converts a set of routines written in ISL into pure B5500 ALGØL in a form appropriate for the final compiler or translator.

An example of the use of semantic tests is the call @ T10 in line 14 of the TWINKLE input of Appendix A. This shows up at word 37 (among others) of PRØTAB (Appendix B). This semantic test routine checks a symbol table and returns true if the identifier just scanned has been declared integer and false otherwise. The value of this test is seen in FPL productions 37 and 42 (Appendix C) which have identical parser symbols. An examination of PRØTAB shows that each can be followed by "←" and an arithmetic expression of arbitrary length are thus are not differentiable by lookahead. A combination of these FPL productions would also fail ultimately because a <boolean expression> (PRØTAB 48) and an <arithmetic expression> (PRØTAB 39) can be syntactically identical (e.g., a single identifier). The syntax of DEMALGL as written here is, in fact, ambiguous without the use of a semantic test.

8.3 Other Considerations

There are some other features of the total system which do not fall under the specific headings of either syntax or semantics. Probably the most important of these is the outstanding automatic error recovery available [22]. During the CF to FPL conversion numerous table entries (in the form of bit patterns) are generated specifying the sets

of valid symbols which may follow in certain situations. There are used in a variety of ways to do what amounts to error correction when no production in a group applies.

The basic error recovery makes use of the information in the marker stack. The approach is to scan ahead until a symbol is found which can validly follow the nonterminal symbol in the process of being recognized. All intervening symbols are then arbitrarily reduced to that symbol. The process is actually slightly more complicated, but it is worthy of note that none of the systems considered in chapter 7 have enough information available at parse time to do anything along these lines.

It should be noted that the final result of inputing syntax and semantics to the appropriate preprocessors is a compilable Burrough's B5500 ALGØL program. This is accomplished by merging the output of the semantics preprocessor with various prewritten pieces of ALGØL code (including a scanner which may be user modified) and possibly the stream of ALGØL procedure calls output by the syntax preprocessor.

The system has been used to generate several translators and parsers. Among these are the TWINKLE and ISL translators and a parser for a large subset of PL/1. Also, presently underway is the generation of a translator for an operating system language, ØSL, which is larger and more complex than ALGØL . In addition to its obvious use as a translator writing system, the system can be used as an aid in writing any program whose operation depends on the syntactic structure of its input. The system has been used in this manner to generate a set of diagnostic programs which have aided in the development of the ILLIAC IV computer

and to generate a symbolic differential equation solver. Other projects of this nature might include a program for reformatting source codes for ALGØL (or other high level language).

The program which does phase two of the syntax preprocessing (i.e., CF to FPL conversion) is a B5500 ALGØL program of approximately 2300 cards in length. As noted, execution time on the sample language DEMALGL was 76 seconds. This time varies greatly as a function of the size and complexity of the input grammar. Although no firm relationship has been established, it appears to increase somewhat more than linearly as the size of PRØTAB increases.

Appendix D is a set of Flow Charts describing the structure of the conversion program with the necessary parts of the TWINKLE compilation included. All methods of internal data representation and many other details have been omitted since they are highly machine dependent.

8.4 Summary

This chapter has briefly described the implemented TWS which uses the CF to FPL conversion algorithm of chapters 3 through 5 as the essential part of its syntax processing phase. In particular, it describes several important features of the system which exist only because the conversion algorithm was specifically designed and implemented to allow them.

The most important of these features is the method of associating semantics with syntax. Specifically mentioned by several users of the system as a significant advantage is the ability to put semantic calls in the grammar at positions other than the end of RHS's. Also

important in this area is the fact that no semantic routine will be executed erroneously. This is possible with many other parsing algorithms which require backing up to a previous point in the parse.

Other features are the use of semantic tests to greatly increase the class of convertible languages, the fact that ambiguous grammars are recognized, the existence of marker symbol information for error recovery and the use of "any" symbols in the input grammar. In addition, the nature of the conversion algorithm and the form of the resulting parser (i.e., FPL productions) make possible optimizations such as the inline use of T (π, n) productions which increase parsing efficiency by 25 to 30 per cent.

9. CONCLUSION

The introduction (chapter 1) of this work sets forth several goals for a translator writing system. These included the construction of translators which are fast, occupy as little space as possible and are capable of generating efficient code when the translator is to be a compiler. Other criteria of importance were simplicity of use and successful processing of as large a class of grammars as possible.

Let us first consider the speed and size of the translators which result from the implements TWS which uses the CF to FPL conversion algorithm of chapters 3 through 5 as the basis for its syntactic preprocessing. It must first be noted that the larger part of any translator or compiler is the semantics and code generation. The TWS's known to this author, leave this to the language designer (i.e., the TWS user). For this reasons we will consider only the speed and size of the parsing phase of translation. Chapter 7 shows that in most, if not all, cases the parsers generated by this method out-perform parsers of a similar nature generated automatically by the other methods known to this writer. In addition to the factors considered in chapter 7, there is the 25 to 30 per cent reduction in parse time mentioned in chapter 8 which results from optimizations most of which are possible only because of the form of the parsers generated.

Of the methods which generate this type of parser, that of DeRemer [15] compares most favorably with the CF to FPL conversion method. Horning and Lalonde [23] have done comparison of parsers generated by the DeRemer method as opposed to parsers based on precedence relationships generated by the methods of McKeeman [4] and Wirth and

and Weber [5]. This empirical study shows the DeRemer parsers, and therefore, by implication, the CF to FPL conversion generated parsers, to be significantly more efficient in both speed and size.

Since the precedence methods mentioned above are, in general, considered practical as the bases for translator writing systems, we must conclude that we have more than adequately met our goals of speed and size.

The next question to be considered is the class of languages which can be successfully processed by the conversion algorithm. For reasons given in chapter 1, the goal was to accept all LR (k) grammars. As pointed out in chapter 6, we have failed to meet this criteria. However, we have, as was also noted in chapter 6, approached LR (k) acceptance to such a great degree as to make this failing relatively insignificant.

The important difference between LR (k) langauges and LR (k) grammars should be noted. For a language to be LR (k) means that there exists an LR (k) grammar for that language. It is possible, and sometimes the case, that a non-LR (k) grammar is written for an LR (k) language. The most frequent reason that a grammar is found unacceptable, however, is that it specifies a language other than the one actually intended. Usually this is an ambiguous grammar.

This brings us to the most serious problem of the implemented TWS. This is the difficulty in determining what changes are necessary in the input grammar when the conversion algorithm does not apply. This problem also occurs in all the other parsing algorithm generators previously discussed but is aggravated to some degree by the intermediate translation from TWINKLE to CF productions. Efforts are presently under

way in an attempt to alleviate this problem. The first such effort is an attempt to determine precisely what information should be made available to the language designer when an unresolvable preclusion is recognized. To date this includes the descriptors of the FPL productions involved and the common multisymbol lookahead string which may follow these conflicting productions. The second such effort is a search for (and hopefully documentation of) a semi-algorithmic approach which the language designer can use to determine what changes are necessary.

This leads us to a consideration of the relative simplicity of using the TWS. With the exception just mentioned and one other, the system appears to be quite good in this area. Of particular value are the flexibility of access to semantic routines (both actions and tests), the automatic error recovery, the recognition of ambiguities and the TWINKLE metalanguage with its "any" symbol constructs and documentation-aiding English forms.

The other exception referred to above is the failure to automate semantics and code generation. These tasks have been simplified through the introduction of the ISL semantics language but it is this author's opinion that no TWS will be entirely satisfactory until the semantic phase of compilation has been automated to a degree at least approaching that already available for the syntactic phase. This failure to automate semantics and, more importantly, code generation does, however, have one advantage: it allows the TWS a broader range of application by not restricting it to the generation of compilers. It also puts no inherent restrictions on the efficiency of the object codes generated by TWS built compilers.

In general, we believe that the TWS described in this work and particularly the CF to FPL conversion algorithm on which it is based are quite useful and practical contributions to the field of language processing.

APPENDIX A

TWINKLE SYNTAX OF DEMAIGL

DEMALGI /SYNTAX

```
A <PROGRAM> IS DEFINED TO BE A <BLOCK> @S 1 ;

A <BLOCK> CONSISTS OF #BEGIN @S3 FOLLOWED BY A POSSIBLY EMPTY
LIST OF [<DECLARATION>  FOLLOWED BY <COMMENT> ]
FOLLOWED BY A LIST OF <STATEMENT>S SEPARATED BY <COMMENT>S
FOLLOWED BY #END @S2 ;

A <DECLARATION> CONSISTS OF
[ #INTEGER @S4 / #BOOLEAN @S5 / #LABEL @S6 ]
FOLLOWED BY A STRING OF [<*I> @S7 ] SEPARATED BY #, ;

A <STATEMENT> IS DEFINED TO BE [ <LABEL> @S8 #: ]* FOLLOWED BY
[ [ #GO #TO OR #GO OR #GOTO ] FOLLOWED BY A <LABEL> @S9 OR
<*I> @T10 [#: #= OR #+ ] <ARITHMETIC EXPRESSION> @S11 OR
<*I> [ #: #= OR #+ ] <BOOLEAN EXPRESSION> @S12 OR
#IF <BOOLEAN EXPRESSION> @S13 #THEN <STATEMENT> @S14 #ELSE <STATEMENT>
@S15 OR EMPTY ] ;

A <COMMENT> CONSISTS OF #; OR #COMMENT FOLLOWED BY A POSSIBLY
EMPTY LIST OF <ANYTHING BUT SEMICOLON>S FOLLOWED BY #; ;

A <LABEL> IS DEFINED TO BE AN <*I> ;

AN <ARITHMETIC EXPRESSION> CONSISTS OF A <TERM> FOLLOWED BY A POSSIBLY
EMPTY LIST OF [[ #+ OR #- ] FOLLOWED BY A <TERM> @S16 ] ;

A <TERM> CONSISTS OF AN <ARITHMETIC PRIMARY> FOLLOWED BY A POSSIBLY
EMPTY LIST OF [[ #x OR #/ ] FOLLOWED BY AN <ARITHMETIC PRIMARY> @S16];
```

```
AN <ARITHMETIC PRIMARY> CONSISTS OF #( <ARITHMETIC EXPRESSION>.#) OR
AN <*I>@T10 @S17 OR A <*N> @S18 ;

A <BOOLEAN EXPRESSION> CONSISTS OF A <BOOLEAN TERM> FOLLOWED BY A
POSSIBLY EMPTY
LIST OF [ #OR FOLLOWED BY A <BOOLEAN TERM> @S16 ];

A <BOOLEAN TERM> IS DEFINED TO BE A <BOOLEAN PRIMARY> FOLLOWED BY A
POSSIBLY EMPTY LIST OF [ #AND <BOOLEAN PRIMARY> @S16 ] ;

A <BOOLEAN PRIMARY> CONSISTS OF #( <BOOLEAN EXPRESSION> #) OR
AN <ARITHMETIC EXPRESSION> FOLLOWED BY [ #= / #≠ / #> / #< / #< / #≥]
FOLLOWED BY AN <ARITHMETIC EXPRESSION> OR AN <*I> @S19 OR
#TRUE @S20 OR #FALSE @S21 ;

AN <ANYTHING BUT SEMICOLON> CONSISTS OF <*I> / <*N> / <*S> / #BEGIN /
#END / #GO / #TO / # GOTO / #COMMENT / #INTEGER / #BOOLEAN / #LABEL /
#IF / #THEN / #ELSE / #AND / #OR / #TRUE /#FALSE /#, / #; / ## / #@ /
#. ;
```

APPENDIX B

PRØTAB OF DEMAIGL

```
COMPLETE PROTAB IS:

     LOC.  LEFT HAND SIDE                                    RIGHT HAND SIDE

      1:  <WHOLE PROGRAM                        >  ::=   EOF MARK
      2:                                                <PROGRAM                            >
      3:                                                 EOF MARK
      4:  <PROGRAM                              >  ::=  <BLOCK DUMMY 2                      >
      5:                                                #END
      6:                                                @S-2
      7:                                                @S-1
      8:  <COMMENT                              >  ::=   ";"
      9:  <COMMENT                              >  ::=  #COMMENT
     10:                                                 ";"
     11:  <COMMENT                              >  ::=  <COMMENT DUMMY 5                    >
     12:                                                 ";"
     13:  <BLOCK DUMMY 1                        >  ::=  <BLOCK DUMMY 1                      >
     14:                                                <DECLARATION DUMMY 3                >
     15:                                                <COMMENT                            >
     16:  <BLOCK DUMMY 1                        >  ::=  #BEGIN
     17:                                                @S-3
     18:                                                <DECLARATION DUMMY 3                >
     19:                                                <COMMENT                            >
     20:  <STATEMENT                            >  ::=  #GO
     21:                                                #TO
     22:                                                 IDENTIFIER
     23:                                                @S-9
     24:  <STATEMENT                            >  ::=  #GO
     25:                                                 IDENTIFIER
     26:                                                @S-9
     27:  <STATEMENT                            >  ::=  #GOTO
     28:                                                 IDENTIFIER
     29:                                                @S-9
     30:  <STATEMENT                            >  ::=   IDENTIFIER
     31:                                                @T-10
     32:                                                 ":"
     33:                                                 "="
     34:                                                <ARITHMETICEXPRESSION               >
     35:                                                @S-11
     36:  <STATEMENT                            >  ::=   IDENTIFIER
     37:                                                @T-10
     38:                                                 "←"
     39:                                                <ARITHMETICEXPRESSION               >
     40:                                                @S-11
     41:  <STATEMENT                            >  ::=   IDENTIFIER
     42:                                                 ":"
     43:                                                 "="
     44:                                                <BOOLEANEXPRESSION                  >
     45:                                                @S-12
     46:  <STATEMENT                            >  ::=   IDENTIFIER
     47:                                                 "←"
     48:                                                <BOOLEANEXPRESSION                  >
     49:                                                @S-12
     50:  <STATEMENT                            >  ::=  #IF
     51:                                                <BOOLEANEXPRESSION                  >
     52:                                                @S-13
     53:                                                #THEN
     54:                                                <STATEMENT                          >
```

```
                                          @S-14
                                          #ELSE
                                          <STATEMENT                >
                                          @S-15
<STATEMENT                 > ::= <STATEMENT DUMMY 4                 >
                                          #GO
                                          #TO
                                           IDENTIFIER
                                          @S-9
<STATEMENT                 > ::= <STATEMENT DUMMY 4                 >
                                          #GO
                                           IDENTIFIER
                                          @S-9
<STATEMENT                 > ::= <STATEMENT DUMMY 4                 >
                                          #GOTO
                                           IDENTIFIER
                                          @S-9
<STATEMENT                 > ::= <STATEMENT DUMMY 4                 >
                                           IDENTIFIER
                                          @T-10
                                           ":"
                                           "="
                                          <ARITHMETICEXPRESSION     >
                                          @S-11
<STATEMENT                 > ::= <STATEMENT DUMMY 4                 >
                                           IDENTIFIER
                                          @T-10
                                           "←"
                                          <ARITHMETICEXPRESSION     >
                                          @S-11
<STATEMENT                 > ::= <STATEMENT DUMMY 4                 >
                                           IDENTIFIER
                                           ":"
                                           "="
                                          <BOOLEANEXPRESSION        >
                                          @S-12
<STATEMENT                 > ::= <STATEMENT DUMMY 4                 >
                                           IDENTIFIER
                                           "←"
                                          <BOOLEANEXPRESSION        >
                                          @S-12
<STATEMENT                 > ::= <STATEMENT DUMMY 4                 >
                                          #IF
                                          <BOOLEANEXPRESSION        >
                                          @S-13
                                          #THEN
                                          <STATEMENT                >
                                          @S-14
                                          #ELSE
                                          <STATEMENT                >
                                          @S-15
<STATEMENT                 > ::= <STATEMENT DUMMY 4                 >
<STATEMENT                 > ::= #IF
                                          <BOOLEANEXPRESSION        >
                                          @S-13
                                          #THEN
                                          @S-14
                                          #ELSE
                                          <STATEMENT                >
```

```
                                            @S-15
<STATEMENT                          > ::=  #IF
                                            <BOOLEANEXPRESSION                  >
                                            @S-13
                                            #THEN
                                            <STATEMENT                          >
                                            @S-14
                                            #ELSE
                                            @S-15
<STATEMENT                          > ::=  <STATEMENT DUMMY 4                   >
                                            #IF
                                            <BOOLEANEXPRESSION                  >
                                            @S-13
                                            #THEN
                                            @S-14
                                            #ELSE
                                            <STATEMENT                          >
                                            @S-15
<STATEMENT                          > ::=  <STATEMENT DUMMY 4                   >
                                            #IF
                                            <BOOLEANEXPRESSION                  >
                                            @S-13
                                            #THEN
                                            <STATEMENT                          >
                                            @S-14
                                            #ELSE
                                            @S-15
<STATEMENT                          > ::=  #IF
                                            <BOOLEANEXPRESSION                  >
                                            @S-13
                                            #THEN
                                            @S-14
                                            #ELSE
                                            @S-15
<STATEMENT                          > ::=  <STATEMENT DUMMY 4                   >
                                            #IF
                                            <BOOLEANEXPRESSION                  >
                                            @S-13
                                            #THEN
                                            @S-14
                                            #ELSE
                                            @S-15
<BLOCK DUMMY 2                      > ::=  <BLOCK DUMMY 2                       >
                                            <COMMENT                            >
                                            <STATEMENT                          >
<BLOCK DUMMY 2                      > ::=  #BEGIN
                                            @S-3
                                            <STATEMENT                          >
<BLOCK DUMMY 2                      > ::=  <BLOCK DUMMY 1                       >
                                            <STATEMENT                          >
<BLOCK DUMMY 2                      > ::=  <BLOCK DUMMY 2                       >
                                            <COMMENT                            >
<BLOCK DUMMY 2                      > ::=  #BEGIN
                                            @S-3
<BLOCK DUMMY 2                      > ::=  <BLOCK DUMMY 1                       >
<DECLARATION DUMMY 3                > ::=  <DECLARATION DUMMY 3                 >
                                             ","
                                             IDENTIFIER
                                            @S-7
```

```
<DECLARATION DUMMY 3                > ::= #INTEGER
                                          @S=4
                                           IDENTIFIER
                                          @S=7
<DECLARATION DUMMY 3                > ::= #BOOLEAN
                                          @S=5
                                           IDENTIFIER
                                          @S=7
<DECLARATION DUMMY 3                > ::= #LABEL
                                          @S=6
                                           IDENTIFIER
                                          @S=7
<STATEMENT DUMMY 4                  > ::= <STATEMENT DUMMY 4                >
                                           IDENTIFIER
                                          @S=8
                                           ":"
<STATEMENT DUMMY 4                  > ::=  IDENTIFIER
                                          @S=8
                                           ":"
<ARITHMETICEXPRESSION               > ::= <TERM                             >
<ARITHMETICEXPRESSION               > ::= <ARITHMETICEXPRESSION DUMMY 6     >
<BOOLEANEXPRESSION                  > ::= <BOOLEANTERM                      >
<BOOLEANEXPRESSION                  > ::= <BOOLEANEXPRESSION DUMMY 8        >
<ANYTHINGBUTSEMICOLON               > ::=  IDENTIFIER
<ANYTHINGBUTSEMICOLON               > ::=  NUMBER
<ANYTHINGBUTSEMICOLON               > ::=  STRING
<ANYTHINGBUTSEMICOLON               > ::= #BEGIN
<ANYTHINGBUTSEMICOLON               > ::= #END
<ANYTHINGBUTSEMICOLON               > ::= #GO
<ANYTHINGBUTSEMICOLON               > ::= #TO
<ANYTHINGBUTSEMICOLON               > ::= #GOTO
<ANYTHINGBUTSEMICOLON               > ::= #COMMENT
<ANYTHINGBUTSEMICOLON               > ::= #INTEGER
<ANYTHINGBUTSEMICOLON               > ::= #BOOLEAN
<ANYTHINGBUTSEMICOLON               > ::= #LABEL
<ANYTHINGBUTSEMICOLON               > ::= #IF
<ANYTHINGBUTSEMICOLON               > ::= #THEN
<ANYTHINGBUTSEMICOLON               > ::= #ELSE
<ANYTHINGBUTSEMICOLON               > ::= #AND
<ANYTHINGBUTSEMICOLON               > ::= #OR
<ANYTHINGBUTSEMICOLON               > ::= #TRUE
<ANYTHINGBUTSEMICOLON               > ::= #FALSE
<ANYTHINGBUTSEMICOLON               > ::=  ","
<ANYTHINGBUTSEMICOLON               > ::=  ":"
<ANYTHINGBUTSEMICOLON               > ::=  "#"
<ANYTHINGBUTSEMICOLON               > ::=  "@"
<ANYTHINGBUTSEMICOLON               > ::=  "."
<COMMENT DUMMY 5                    > ::= <COMMENT DUMMY 5                  >
                                          <ANYTHINGBUTSEMICOLON             >
<COMMENT DUMMY 5                    > ::= #COMMENT
                                          <ANYTHINGBUTSEMICOLON             >
<TERM                               > ::= <ARITHMETICPRIMARY                >
<TERM                               > ::= <TERM DUMMY 7                     >
<ARITHMETICEXPRESSION DUMMY 6       > ::= <ARITHMETICEXPRESSION DUMMY 6     >
                                           "+"
                                          <TERM                             >
                                          @S=16
<ARITHMETICEXPRESSION DUMMY 6       > ::= <ARITHMETICEXPRESSION DUMMY 6     >
                                           "-"
```

```
                                                 <TERM                                    >
                                                 @S=16
<ARITHMETICEXPRESSION DUMMY 6          > ::=     <TERM                                    >
                                                  "+"
                                                 <TERM                                    >
                                                 @S=16
<ARITHMETICEXPRESSION DUMMY 6          > ::=     <TERM                                    >
                                                  "-"
                                                 <TERM                                    >
                                                 @S=16
<ARITHMETICPRIMARY                     > ::=      "("
                                                 <DUMMY 23                                >
<ARITHMETICPRIMARY                     > ::=      IDENTIFIER
                                                 @T=10
                                                 @S=17
<ARITHMETICPRIMARY                     > ::=      NUMBER
                                                 @S=18
<TERM DUMMY 7                          > ::=     <TERM DUMMY 7                            >
                                                  "x"
                                                 <ARITHMETICPRIMARY                       >
                                                 @S=16
<TERM DUMMY 7                          > ::=     <TERM DUMMY 7                            >
                                                  "/"
                                                 <ARITHMETICPRIMARY                       >
                                                 @S=16
<TERM DUMMY 7                          > ::=     <ARITHMETICPRIMARY                       >
                                                  "x"
                                                 <ARITHMETICPRIMARY                       >
                                                 @S=16
<TERM DUMMY 7                          > ::=     <ARITHMETICPRIMARY                       >
                                                  "/"
                                                 <ARITHMETICPRIMARY                       >
                                                 @S=16
<BOOLEANTERM                           > ::=     <BOOLEANPRIMARY                          >
<BOOLEANTERM                           > ::=     <BOOLEANTERM DUMMY 9                     >
<BOOLEANEXPRESSION DUMMY 8             > ::=     <BOOLEANEXPRESSION DUMMY 8               >
                                                 #OR
                                                 <BOOLEANTERM                             >
                                                 @S=16
<BOOLEANEXPRESSION DUMMY 8             > ::=     <BOOLEANTERM                             >
                                                 #OR
                                                 <BOOLEANTERM                             >
                                                 @S=16
<BOOLEANPRIMARY                        > ::=      "("
                                                 <BOOLEANEXPRESSION                       >
                                                  ")"
<BOOLEANPRIMARY                        > ::=     <ARITHMETICEXPRESSION                    >
                                                  "="
                                                 <ARITHMETICEXPRESSION                    >
<BOOLEANPRIMARY                        > ::=     <ARITHMETICEXPRESSION                    >
                                                  "≠"
                                                 <ARITHMETICEXPRESSION                    >
<BOOLEANPRIMARY                        > ::=     <ARITHMETICEXPRESSION                    >
                                                  ">"
                                                 <ARITHMETICEXPRESSION                    >
<BOOLEANPRIMARY                        > ::=     <ARITHMETICEXPRESSION                    >
                                                  "<"
                                                 <ARITHMETICEXPRESSION                    >
<BOOLEANPRIMARY                        > ::=     <ARITHMETICEXPRESSION                    >
```

```
                                                  "≤"
                                                  <ARITHMETICEXPRESSION        >
<BOOLEANPRIMARY                         > ::= <ARITHMETICEXPRESSION        >
                                                  "≥"
                                                  <ARITHMETICEXPRESSION        >
<BOOLEANPRIMARY                         > ::=  IDENTIFIER
                                                  @S=19
<BOOLEANPRIMARY                         > ::= #TRUE
                                                  @S=20
<BOOLEANPRIMARY                         > ::= #FALSE
                                                  @S=21
<BOOLEANTERM DUMMY 9                    > ::= <BOOLEANTERM DUMMY 9         >
                                                  #AND
                                                  <BOOLEANPRIMARY              >
                                                  @S=16
<BOOLEANTERM DUMMY 9                    > ::= <BOOLEANPRIMARY              >
                                                  #AND
                                                  <BOOLEANPRIMARY              >
                                                  @S=16
<DUMMY 23                               > ::= <ARITHMETICEXPRESSION        >
                                                  ")"
```

APPENDIX C

FPL PRODUCTION SET FOR DEMALGL

THE FOLLOWING ARE THE FLOYD PRODUCTIONS FOR DEMAIGL:

```
NTH NONE  GROUP     0:

          0:      ERROR PRODUCTION:   SKIP SYMBOLS.

START     GROUP     1:

          1:                    PUSH PAR  WHOLE PROGRAM
            : SCAN (EOF);  GO TO    TH   0 .

NTPN   0  GROUP     2:

          2:      GO TO DONEWITHPASS1.

  TH   0  GROUP     3:      WHOLE PROGRAM

          3:      PRODUCTION ASSOCIATED WITH PROTAB NO.:       1.
                  TOP STACK SYMBOL:   EOF
                                PUSH PAR  PROGRAM
                   SCAN;  GO TO    TH   1 .

  TH   1  GROUP     4:      PROGRAM

          4:      PRODUCTION ASSOCIATED WITH PROTAB NO.:      17.
                  TOP STACK SYMBOL:      #BEGIN    @S-3
                    IS INPUT:  ANY=   6
                                PUSH PAR  DECLARATION DUMMY 3
                   SCAN;  GO TO    TH   8 .

          5:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     160.
                  TOP STACK SYMBOL:      #BEGIN    @S-3
                    IS INPUT:  ANY=   7
                                PUSH PAR  STATEMENT
                   SCAN;  GO TO    TH   6 .

          6:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     167.
                  TOP STACK SYMBOL:      #BEGIN    @S-3
                                REDUCE TO BLOCK DUMMY 2
                         GO TO   DNT   7 .
 NTH   1  GROUP     5:      PROGRAM

          7:      PRODUCTION ASSOCIATED WITH PROTAB NO.:       4.
                  TOP STACK SYMBOL:      BLOCK DUMMY 2
                    IS INPUT:     #END
                   SCAN;

                  TOP STACK SYMBOL:      #END      @S-2        @S-1
                                REDUCE TO PROGRAM
                         GO TO   DNT   1 .

          9:      COMBINED PRODUCTION ASSOCIATED WITH PROTAB NO.:
                           156,  164,
                  TOP STACK SYMBOL:      BLOCK DUMMY 2
                                PUSH SPR  COMMENT
                   SCAN;  GO TO    TH   4 .
```

```
          10:     PRODUCTION ASSOCIATED WITH PROTAB NO.:      13.
                  TOP STACK SYMBOL:       BLOCK DUMMY 1
                    IS INPUT:  ANY=   6
                                   PUSH PAR  DECLARATION DUMMY 3
                   SCAN;  GO TO    TH   8 .

          11:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     162.
                  TOP STACK SYMBOL:       BLOCK DUMMY 1
                    IS INPUT:  ANY=   7
                                   PUSH PAR  STATEMENT
                   SCAN;  GO TO    TH   6 .

          12:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     168.
                  TOP STACK SYMBOL:       BLOCK DUMMY 1
                                   REDUCE TO BLOCK DUMMY 2
                          GO TO    DNT   7 .

          13:     ERROR PRODUCTION:   SKIP SYMBOLS.

NTPN    2 GROUP     6:      PROGRAM

          14:     PRODUCTION ASSOCIATED WITH PROTAB NO.:       2.
                  TOP STACK SYMBOL:       PROGRAM
                   SCAN;

                  TOP STACK SYMBOL:   EOF
                                   REDUCE TO WHOLE PROGRAM
                          GO TO    DNT   0 .

  TH    4 GROUP     7:      COMMENT

          16:     PRODUCTION ASSOCIATED WITH PROTAB NO.:       8.
                  TOP STACK SYMBOL:   ";"
                                   REDUCE TO COMMENT
                          GO TO    DNT   4 .

          17:     PRODUCTION ASSOCIATED WITH PROTAB NO.:       9.
                  TOP STACK SYMBOL:      #COMMENT
                    IS INPUT:    ";"
                   SCAN;

                  TOP STACK SYMBOL:   ";"
                                   REDUCE TO COMMENT
                          GO TO    DNT   4 .

          19:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     222.
                  TOP STACK SYMBOL:      #COMMENT
                                   PUSH PAR  ANYTHINGBUTSEMICOLON
                   SCAN;  GO TO    TH   13 .

          20:     ERROR PRODUCTION:   SKIP SYMBOLS.

NTH     4 GROUP     8:      COMMENT

          21:     PRODUCTION ASSOCIATED WITH PROTAB NO.:      11.
                  TOP STACK SYMBOL:       COMMENT DUMMY 5
                    IS INPUT:    ";"
                   SCAN;

                  TOP STACK SYMBOL:   ";"
```

```
                                  REDUCE TO COMMENT
                        GO TO   DNT    4 .

            23:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     220.
                    TOP STACK SYMBOL:       COMMENT DUMMY 5
                                  PUSH BAR  ANYTHINGBUTSEMICOLON
                     SCAN;  GO TO   TH  13 .

            24:     ERROR PRODUCTION:   SKIP SYMBOLS.

NTPN   15   GROUP    9:      COMMENT

            25:     PRODUCTION ASSOCIATED WITH PROTAB NO.:      15.
                    TOP STACK SYMBOL:       COMMENT
                                  REDUCE TO BLOCK DUMMY 1
                        GO TO   DNT    5 .

NTPN   19   GROUP   10:      COMMENT

            26:     PRODUCTION ASSOCIATED WITH PROTAB NO.:      19.
                    TOP STACK SYMBOL:       COMMENT
                                  REDUCE TO BLOCK DUMMY 1
                        GO TO   DNT    5 .

  TH    6   GROUP   11:      STATEMENT

            27:     PRODUCTION ASSOCIATED WITH PROTAB NO.:      20.
                    TOP STACK SYMBOL:      #GO
                      IS INPUT:      #TO
                     SCAN;

                    TOP STACK SYMBOL:      #TO
                     SCAN;

                    TOP STACK SYMBOL:  <*I>    @S-9
                                  REDUCE TO STATEMENT
                        GO TO   DNT    6 .

            30:     PRODUCTION ASSOCIATED WITH PROTAB NO.:      24.
                    TOP STACK SYMBOL:      #GO
                     SCAN;

                    TOP STACK SYMBOL:  <*I>    @S-9
                                  REDUCE TO STATEMENT
                        GO TO   DNT    6 .

            32:     PRODUCTION ASSOCIATED WITH PROTAB NO.:      27.
                    TOP STACK SYMBOL:      #GOTO
                     SCAN;

                    TOP STACK SYMBOL:  <*I>    @S-9
                                  REDUCE TO STATEMENT
                        GO TO   DNT    6 .

            34:     PRODUCTION ASSOCIATED WITH PROTAB NO.:      31.
                    TOP STACK SYMBOL:  <*I>    @T-10
                      IS INPUT:    ":"
                     SCAN;

                    TOP STACK SYMBOL:    ":"
                     SCAN;
```

```
          TOP STACK SYMBOL:   "="
                            PUSH PAR  ARITHMETICEXPRESSION
           SCAN;  GO TO    TH  11 .

    37:   PRODUCTION ASSOCIATED WITH PROTAB NO.:      37.
          TOP STACK SYMBOL:  <*I>    @T=10
           SCAN;

          TOP STACK SYMBOL:   "+"
                            PUSH PAR  ARITHMETICEXPRESSION
           SCAN;  GO TO    TH  11 .

    39:   PRODUCTION ASSOCIATED WITH PROTAB NO.:      41.
          TOP STACK SYMBOL:  <*I>
            IS INPUT:   ":"   "="
           SCAN;

          TOP STACK SYMBOL:   ":"
           SCAN;

          TOP STACK SYMBOL:   "="
                            PUSH PAR  BOOLEANEXPRESSION
           SCAN;  GO TO    TH  12 .

    42:   PRODUCTION ASSOCIATED WITH PROTAB NO.:      46.
          TOP STACK SYMBOL:  <*I>
            IS INPUT:   "+"
           SCAN;

          TOP STACK SYMBOL:   "+"
                            PUSH PAR  BOOLEANEXPRESSION
           SCAN;  GO TO    TH  12 .

    44:   PRODUCTION ASSOCIATED WITH PROTAB NO.:     190.
          TOP STACK SYMBOL:  <*I>    @S=B
           SCAN;

          TOP STACK SYMBOL:   ":"
                            REDUCE TO STATEMENT DUMMY 4
                 GO TO    DNT  10 .

    46:   COMBINED PRODUCTION ASSOCIATED WITH PROTAB NO.:
                      50,  107,  115,  141,
          TOP STACK SYMBOL:     #IF
                            PUSH SPR  BOOLEANEXPRESSION
           SCAN;  GO TO    TH  12 .

    47:   ERROR PRODUCTION:   SKIP SYMBOLS.

NTH   6  GROUP  12:     STATEMENT

    48:   COMBINED PRODUCTION ASSOCIATED WITH PROTAB NO.:
                     59,   60,
          TOP STACK SYMBOL:       STATEMENT DUMMY 4
            IS INPUT:      #GO
           SCAN;  GO TO  CTPN   2 .

    49:   PRODUCTION ASSOCIATED WITH PROTAB NO.:      68.
          TOP STACK SYMBOL:       STATEMENT DUMMY 4
            IS INPUT:      #GOTO
```

```
                    SCAN;

                   TOP STACK SYMBOL:      #GOTO
                    SCAN;

                   TOP STACK SYMBOL:  <*I>    #S-9
                                  REDUCE TO STATEMENT
                         GO TO    DNT    6 .

          52:      COMBINED PRODUCTION ASSOCIATED WITH PROTAB NO.:
                           72,   79,   85,   91,  185,
                   TOP STACK SYMBOL:        STATEMENT DUMMY 4
                     IS INPUT:  <*I>
                    SCAN;  GO TO  CTPN    3 .

          53:      COMBINED PRODUCTION ASSOCIATED WITH PROTAB NO.:
                           96,  123,  132,  148,
                   TOP STACK SYMBOL:        STATEMENT DUMMY 4
                     IS INPUT:       #IF
                    SCAN:  GO TO  CTPN    4 .
          54:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     106.
                   TOP STACK SYMBOL:        STATEMENT DUMMY 4
                                  REDUCE TO STATEMENT
                         GO TO    DNT    6 .

          55:      ERROR PRODUCTION:   SKIP SYMBOLS.

NTPN  54  GROUP    13:      STATEMENT

          56:      PRODUCTION ASSOCIATED WITH PROTAB NO.:      55.
                   TOP STACK SYMBOL:        STATEMENT       #S-14
                    SCAN;

                   TOP STACK SYMBOL:      #ELSE
                                  PUSH PAR  STATEMENT
                    SCAN:  GO TO     TH    6 .

NTPN  57  GROUP    14:      STATEMENT

          58:      PRODUCTION ASSOCIATED WITH PROTAB NO.:      58.
                   TOP STACK SYMBOL:        STATEMENT       #S-15
                                  REDUCE TO STATEMENT
                         GO TO    DNT    6 .

NTPN 101  GROUP    15:      STATEMENT

          59:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     102.
                   TOP STACK SYMBOL:        STATEMENT       #S-14
                    SCAN:

                   TOP STACK SYMBOL:      #ELSE
                                  PUSH PAR  STATEMENT
                    SCAN:  GO TO     TH    6 .

NTPN 104  GROUP    16:      STATEMENT

          61:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     105.
                   TOP STACK SYMBOL:        STATEMENT       #S-15
                                  REDUCE TO STATEMENT
                         GO TO    DNT    6 .
```

```
NTPN 113  GROUP   17:      STATEMENT

          62:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     114.
                  TOP STACK SYMBOL:      STATEMENT        @S-15
                                 REDUCE TO STATEMENT
                         GO TO   DNT   6 .

NTPN 119  GROUP   18:      STATEMENT

          63:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     120.
                  TOP STACK SYMBOL:      STATEMENT        @S-14
                   SCAN:

                  TOP STACK SYMBOL:    #ELSE     @S-15
                                 REDUCE TO STATEMENT
                         GO TO   DNT   6 .

NTPN 130  GROUP   19:      STATEMENT

          65:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     131.
                  TOP STACK SYMBOL:      STATEMENT        @S-15
                                 REDUCE TO STATEMENT
                         GO TO   DNT   6 .

NTPN 137  GROUP   20:      STATEMENT

          66:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     138.
                  TOP STACK SYMBOL:      STATEMENT        @S-14
                   SCAN:

                  TOP STACK SYMBOL:    #ELSE     @S-15
                                 REDUCE TO STATEMENT
                         GO TO   DNT   6 .

NTPN 158  GROUP   21:      STATEMENT

          68:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     158.
                  TOP STACK SYMBOL:      STATEMENT
                                 REDUCE TO BLOCK DUMMY 2
                         GO TO   DNT   7 .

NTPN 161  GROUP   22:      STATEMENT

          69:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     161.
                  TOP STACK SYMBOL:      STATEMENT
                                 REDUCE TO BLOCK DUMMY 2
                         GO TO   DNT   7 .

NTPN 163  GROUP   23:      STATEMENT

          70:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     163.
                  TOP STACK SYMBOL:      STATEMENT
                                 REDUCE TO BLOCK DUMMY 2
                         GO TO   DNT   7 .

  TF    8  GROUP   24:      DECLARATION DUMMY 3

          71:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     174.
                  TOP STACK SYMBOL:    #INTEGER          @S-4
                   SCAN:
```

```
                    TOP STACK SYMBOL:  <*I>   @S-7
                                      REDUCE TO DECLARATION DUMMY 3
                          GO TO   DNT   8 .

          73:       PRODUCTION ASSOCIATED WITH PROTAB NO.:     178.
                    TOP STACK SYMBOL:      #BOOLEAN        @S-5
                     SCAN;

                    TOP STACK SYMBOL:  <*I>   @S-7
                                      REDUCE TO DECLARATION DUMMY 3
                          GO TO   DNT   8 .

          75:       PRODUCTION ASSOCIATED WITH PROTAB NO.:     182.
                    TOP STACK SYMBOL:      #LABEL    @S-6
                     SCAN;

                    TOP STACK SYMBOL:  <*I>   @S-7
                                      REDUCE TO DECLARATION DUMMY 3
                          GO TO   DNT   8 .

          77:       ERROR PRODUCTION:   SKIP SYMBOLS.

NTEN  14  GROUP     25:     DECLARATION DUMMY 3

          78:       PRODUCTION ASSOCIATED WITH PROTAB NO.:     169.
                    TOP STACK SYMBOL:       DECLARATION DUMMY
                      IS INPUT:   ","
                     SCAN;

                    TOP STACK SYMBOL:    ","
                     SCAN;

                    TOP STACK SYMBOL:  <*I>   @S-7
                                      REDUCE TO DECLARATION DUMMY 3
                          GO TO   DNT   8 .

          81:       PRODUCTION ASSOCIATED WITH PROTAB NO.:      14.
                    TOP STACK SYMBOL:       DECLARATION DUMMY
                                      PUSH PAR  COMMENT
                     SCAN:  GO TO     TH   4 .

NTEN  18  GROUP     26:     DECLARATION DUMMY 3

          82:       PRODUCTION ASSOCIATED WITH PROTAB NO.:     169.
                    TOP STACK SYMBOL:       DECLARATION DUMMY
                      IS INPUT:   ","
                     SCAN;

                    TOP STACK SYMBOL:    ","
                     SCAN;

                    TOP STACK SYMBOL:  <*I>   @S-7
                                      REDUCE TO DECLARATION DUMMY 3
                          GO TO   DNT   8 .

          85:       PRODUCTION ASSOCIATED WITH PROTAB NO.:      18.
                    TOP STACK SYMBOL:       DECLARATION DUMMY
                                      PUSH PAR  COMMENT
                     SCAN:  GO TO     TH   4 .

 TH   11  GROUP     27:     ARITHMETICEXPRESSION
```

```
         86:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     242.
                 TOP STACK SYMBOL:   "("
                                       PUSH PAR  DUMMY 23
                  SCAN:  GO TO       TH  23 .

         87:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     246.
                 TOP STACK SYMBOL:  <*I>   @T=10        @S=17
                                       REDUCE TO ARITHMETICPRIMARY
                         GO TO       DNT  17 .

         88:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     248.
                 TOP STACK SYMBOL:  <*N>   @S=18
                                       REDUCE TO ARITHMETICPRIMARY
                         GO TO       DNT  17 .

         89:     ERROR PRODUCTION:   SKIP SYMBOLS.

NTH  11  GROUP   28:     ARITHMETICEXPRESSION

         90:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     192.
                 TOP STACK SYMBOL:        TERM
                   IS INPUT:  ANY-   8
                   IS INPUT:      #AND
                   IS INPUT:      #OR
                   IS INPUT:      #ELSE
                   IS INPUT:      #THEN
                   IS INPUT:      #END
                   IS INPUT:    "="
                   IS INPUT:    "≠"
                   IS INPUT:    "≤"
                   IS INPUT:    ")"
                   IS INPUT:    "<"
                   IS INPUT:    "≥"
                   IS INPUT:    ">"
                                       REDUCE TO ARITHMETICEXPRESSION
                         GO TO       DNT  11 .

         91:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     234.
                 TOP STACK SYMBOL:        TERM
                   IS INPUT:    "+"
                  SCAN:

                 TOP STACK SYMBOL:   "+"
                                       PUSH PAR  TERM
                  SCAN:  GO TO       TH  15 .

         93:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     238.
                 TOP STACK SYMBOL:        TERM
                  SCAN:

                 TOP STACK SYMBOL:   "-"
                                       PUSH PAR  TERM
                  SCAN:  GO TO       TH  15 .

         95:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     193.
                 TOP STACK SYMBOL:        ARITHMETICEXPRESST
                   IS INPUT:  ANY-   8
                   IS INPUT:      #AND
                   IS INPUT:      #OR
                   IS INPUT:      #ELSE
```

```
          IS INPUT:      #THEN
          IS INPUT:      #END
          IS INPUT:   "="
          IS INPUT:   "≠"
          IS INPUT:   "≤"
          IS INPUT:   ")"
          IS INPUT:   "<"
          IS INPUT:   "≥"
          IS INPUT:   ">"
                         REDUCE TO ARITHMETICEXPRESSION
               GO TO     DNT  11 .

96:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     226.
        TOP STACK SYMBOL:        ARITHMETICEXPRESSI
          IS INPUT:   "+"
         SCAN:

        TOP STACK SYMBOL:    "+"
                         PUSH PAR  TERM
         SCAN:  GO TO     TH  15 .

98:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     230.
        TOP STACK SYMBOL:        ARITHMETICEXPRESSI
         SCAN:

        TOP STACK SYMBOL:    "-"
                         PUSH PAR  TERM
         SCAN:  GO TO     TH  15 .

100:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     224.
        TOP STACK SYMBOL:        ARITHMETICPRIMARY
          IS INPUT:  ANY-   8
          IS INPUT:      #AND
          IS INPUT:      #OR
          IS INPUT:      #ELSE
          IS INPUT:      #THEN
          IS INPUT:      #END
          IS INPUT:   "="
          IS INPUT:   "≠"
          IS INPUT:   "≤"
          IS INPUT:   ")"
          IS INPUT:   "-"
          IS INPUT:   "<"
          IS INPUT:   "+"
          IS INPUT:   "≥"
          IS INPUT:   ">"
                         REDUCE TO TERM
               GO TO     DNT  15 .

101:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     257.
        TOP STACK SYMBOL:        ARITHMETICPRIMARY
          IS INPUT:   "x"
         SCAN:

        TOP STACK SYMBOL:    "x"
                         PUSH PAR  ARITHMETICPRIMARY
         SCAN:  GO TO     TH  17 .

103:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     261.
        TOP STACK SYMBOL:        ARITHMETICPRIMARY
         SCAN:
```

```
                    TOP STACK SYMBOL:    "/"
                                   PUSH PAR  ARITHMETICPRIMARY
                     SCAN;  GO TO    TH  17 .

          105:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     225.
                    TOP STACK SYMBOL:       TERM DUMMY 7
                      IS INPUT:  ANY=   8
                      IS INPUT:      #AND
                      IS INPUT:      #OR
                      IS INPUT:      #ELSE
                      IS INPUT:      #THEN
                      IS INPUT:      #END
                      IS INPUT:    "="
                      IS INPUT:    "≠"
                      IS INPUT:    "≤"
                      IS INPUT:    ")"
                      IS INPUT:    "="
                      IS INPUT:    "≤"
                      IS INPUT:    "≥"
                      IS INPUT:    ">"
                                   REDUCE TO TERM
                            GO TO   DNT  15 .

          106:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     249.
                    TOP STACK SYMBOL:       TERM DUMMY 7
                      IS INPUT:    "x"
                     SCAN;

                    TOP STACK SYMBOL:    "x"
                                   PUSH PAR  ARITHMETICPRIMARY
                     SCAN;  GO TO    TH  17 .

          108:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     253.
                    TOP STACK SYMBOL:       TERM DUMMY 7
                     SCAN;

                    TOP STACK SYMBOL:    "/"
                                   PUSH PAR  ARITHMETICPRIMARY
                     SCAN;  GO TO    TH  17 .

          110:      ERROR PRODUCTION:   SKIP SYMBOLS.

NTRM   34  GROUP    29:       ARITHMETICEXPRESSION

          111:      PRODUCTION ASSOCIATED WITH PROTAB NO.:      35.
                    TOP STACK SYMBOL:       ARITHMETICEXPRESSI   @S-11
                                   REDUCE TO STATEMENT
                            GO TO   DNT   6 .

NTRM   39  GROUP    30:       ARITHMETICEXPRESSION

          112:      PRODUCTION ASSOCIATED WITH PROTAB NO.:      40.
                    TOP STACK SYMBOL:       ARITHMETICEXPRESSI   @S-11
                                   REDUCE TO STATEMENT
                            GO TO   DNT   6 .

NTRM   77  GROUP    31:       ARITHMETICEXPRESSION

          113:      PRODUCTION ASSOCIATED WITH PROTAB NO.:      78.
                    TOP STACK SYMBOL:       ARITHMETICEXPRESSI   @S-11
```

```
                                          PEDUCF TO STATFMENT
                                  GO TO   DNT   6 .

NTPN  83  GROUP    32:      ARITHMETICEXPRESSION

         114:      PRODUCTION ASSOCIATED WITH PROTAB NO.:       84.
                   TOP STACK SYMBOL:       ARITHMETICEXPRESSI   PS-11
                                  REDUCE TO STATEMENT
                          GO TO   DNT   6 .

NTPN 280  GROUP    33:      ARITHMETICEXPRESSION

         115:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     280.
                   TOP STACK SYMBOL:       ARITHMETICEXPRESSI
                                  REDUCE TO BOOLEANPRIMARY
                          GO TO   DNT  21 .

NTPN 283  GROUP    34:      ARITHMETICEXPRESSION

         116:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     283.
                   TOP STACK SYMBOL:       ARITHMETICEXPRESSI
                                  REDUCE TO BOOLEANPRIMARY
                          GO TO   DNT  21 .

NTPN 286  GROUP    35:      ARITHMETICEXPRESSION

         117:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     286.
                   TOP STACK SYMBOL:       ARITHMETICEXPRESSI
                                  REDUCE TO BOOLEANPRIMARY
                          GO TO   DNT  21 .

NTPN 289  GROUP    36:      ARITHMETICEXPRESSION

         118:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     289.
                   TOP STACK SYMBOL:       ARITHMETICEXPRESSI
                                  REDUCE TO BOOLEANPRIMARY
                          GO TO   DNT  21 .
NTPN 292  GROUP    37:      ARITHMETICEXPRESSION

         119:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     292.
                   TOP STACK SYMBOL:       ARITHMETICEXPRESSI
                                  REDUCE TO BOOLEANPRIMARY
                          GO TO   DNT  21 .

NTPN 295  GROUP    38:      ARITHMETICEXPRESSION

         120:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     295.
                   TOP STACK SYMBOL:       ARITHMETICEXPRESSI
                                  REDUCE TO BOOLEANPRIMARY
                          GO TO   DNT  21 .

  TH  12  GROUP    39:      BOOLEANEXPRESSION

         121:      COMBINED PRODUCTION ASSOCIATED WITH PROTAB NO.:
                               242,  275,
                   TOP STACK SYMBOL:   "("
                                  PUSH FOR   COMBINED GROUP NO.    5
                    SCAN:  GO TO   CTH    5 .

         122:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     246.
                   TOP STACK SYMBOL:  <*I>    BT-10        PS-17
```

```
                                REDUCE TO ARITHMETICPRIMARY
                      GO TO     DNT  17 .

          123:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     297.
                   TOP STACK SYMBOL:  <*I>    @S-19
                                REDUCE TO BOOLEANPRIMARY
                      GO TO     DNT  21 .

          124:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     298.
                   TOP STACK SYMBOL:  <*N>    @S-18
                                REDUCE TO ARITHMETICPRIMARY
                      GO TO     DNT  17 .

          125:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     299.
                   TOP STACK SYMBOL:      #TRUE     @S-20
                                REDUCE TO BOOLEANPRIMARY
                      GO TO     DNT  21 .

          126:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     301.
                   TOP STACK SYMBOL:      #FALSE    @S-21
                                REDUCE TO BOOLEANPRIMARY
                      GO TO     DNT  21 .

          127:     ERROR PRODUCTION:   SKIP SYMBOLS.

NTH  12  GROUP     40:      BOOLEANEXPRESSION

          128:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     192.
                   TOP STACK SYMBOL:        TERM
                     IS INPUT:  ANY-   A
                     IS INPUT:      #AND
                     IS INPUT:      #OR
                     IS INPUT:      #ELSE
                     IS INPUT:      #THEN
                     IS INPUT:      #END
                     IS INPUT:    "="
                     IS INPUT:    "≠"
                     IS INPUT:    "<"
                     IS INPUT:    ")"
                     IS INPUT:    "≤"
                     IS INPUT:    ">"
                                REDUCE TO ARITHMETICEXPRESSION
                      GO TO     DNT  11 .

          129:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     234.
                   TOP STACK SYMBOL:        TERM
                     IS INPUT:    "+"
                    SCAN;

                   TOP STACK SYMBOL:    "+"
                                PUSH BAR  TERM
                    SCAN;  GO TO     TH  15 .

          131:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     238.
                   TOP STACK SYMBOL:        TERM
                    SCAN;

                   TOP STACK SYMBOL:    "-"
                                PUSH BAR  TERM
                    SCAN;  GO TO     TH  15 .
```

```
133:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     193.
         TOP STACK SYMBOL:        ARITHMETICEXPRESSI
           IS INPUT:  ANY=   8
           IS INPUT:      #AND
           IS INPUT:      #OR
           IS INPUT:      #ELSE
           IS INPUT:      #THEN
           IS INPUT:      #END
           IS INPUT:    "="
           IS INPUT:    "≠"
           IS INPUT:    "≤"
           IS INPUT:    ")"
           IS INPUT:    "<"
           IS INPUT:    "≥"
           IS INPUT:    ">"
                          REDUCE TO ARITHMETICEXPRESSION
                 GO TO    DNT  11 .

134:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     226.
         TOP STACK SYMBOL:        ARITHMETICEXPRESSI
           IS INPUT:    "+"
          SCAN;
         TOP STACK SYMBOL:   "+"
                          PUSH PAR  TERM
          SCAN:  GO TO     TH  15 .

136:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     230.
         TOP STACK SYMBOL:        ARITHMETICEXPRESSI
          SCAN;

         TOP STACK SYMBOL:   "-"
                          PUSH PAR  TERM
          SCAN:  GO TO     TH  15 .

138:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     194.
         TOP STACK SYMBOL:        BOOLEANTERM
           IS INPUT:  ANY=   8
           IS INPUT:      #ELSE
           IS INPUT:      #THEN
           IS INPUT:      #END
           IS INPUT:    ")"
                          REDUCE TO BOOLEANEXPRESSION
                 GO TO    DNT  12 .

139:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     271.
         TOP STACK SYMBOL:        BOOLEANTERM
          SCAN:

         TOP STACK SYMBOL:      #OR
                          PUSH PAR  BOOLEANTERM
          SCAN:  GO TO     TH  19 .

141:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     195.
         TOP STACK SYMBOL:        BOOLEANEXPRESSION
           IS INPUT:  ANY=   8
           IS INPUT:      #ELSE
           IS INPUT:      #THEN
           IS INPUT:      #END
           IS INPUT:    ")"
                          REDUCE TO BOOLEANEXPRESSION
                 GO TO    DNT  12 .
```

```
142:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     267.
         TOP STACK SYMBOL:        BOOLEANEXPRESSION
          SCAN;

         TOP STACK SYMBOL:       #OR
                        PUSH PAR  BOOLEANTERM
          SCAN;  GO TO    TH  19 .

144:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     224.
         TOP STACK SYMBOL:        ARITHMETICPRIMARY
           IS INPUT:  ANY-   8
           IS INPUT:      #AND
           IS INPUT:      #OR
           IS INPUT:      #ELSE
           IS INPUT:      #THEN
           IS INPUT:      #END
           IS INPUT:    "="
           IS INPUT:    "≠"
           IS INPUT:    "≤"
           IS INPUT:    ")"
           IS INPUT:    "-"
           IS INPUT:    "<"
           IS INPUT:    "+"
           IS INPUT:    "≥"
           IS INPUT:    ">"
                        REDUCE TO TERM
               GO TO    DNT  15 .

145:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     257.
         TOP STACK SYMBOL:        ARITHMETICPRIMARY
           IS INPUT:    "x"
          SCAN;

         TOP STACK SYMBOL:    "x"
                        PUSH PAR  ARITHMETICPRIMARY
          SCAN;  GO TO    TH  17 .

147:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     261.
         TOP STACK SYMBOL:        ARITHMETICPRIMARY
          SCAN;

         TOP STACK SYMBOL:    "/"
                        PUSH PAR  ARITHMETICPRIMARY
          SCAN;  GO TO    TH  17 .

149:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     225.
         TOP STACK SYMBOL:        TERM DUMMY 7
           IS INPUT:  ANY-   8
           IS INPUT:      #AND
           IS INPUT:      #OR
           IS INPUT:      #ELSE
           IS INPUT:      #THEN
           IS INPUT:      #END
           IS INPUT:    "="
           IS INPUT:    "≠"
           IS INPUT:    "≤"
           IS INPUT:    ")"
           IS INPUT:    "-"
           IS INPUT:    "<"
           IS INPUT:    "+"
```

```
           IS INPUT:   "≥"
           IS INPUT:   ">"
                          REDUCE TO TERM
                GO TO     DNT  15 .

150:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     249.
         TOP STACK SYMBOL:        TERM DUMMY 7
           IS INPUT:   "×"
          SCAN;

         TOP STACK SYMBOL:    "×"
                          PUSH PAR  ARITHMETICPRIMARY
          SCAN;  GO TO      TH  17 .

152:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     253.
         TOP STACK SYMBOL:        TERM DUMMY 7
          SCAN;

         TOP STACK SYMBOL:    "/"
                          PUSH PAR  ARITHMETICPRIMARY
          SCAN;  GO TO      TH  17 .

154:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     265.
         TOP STACK SYMBOL:        BOOLEANPRIMARY
           IS INPUT:  ANY-   8
           IS INPUT:      #OR
           IS INPUT:      #ELSE
           IS INPUT:      #THEN
           IS INPUT:      #END
           IS INPUT:   ")"
                          REDUCE TO BOOLEANTERM
                GO TO     DNT  19 .

155:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     306.
         TOP STACK SYMBOL:        BOOLEANPRIMARY
          SCAN;

         TOP STACK SYMBOL:      #AND
                          PUSH PAR  BOOLEANPRIMARY
          SCAN;  GO TO      TH  21 .

157:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     266.
         TOP STACK SYMBOL:        BOOLEANTERM DUMMY
           IS INPUT:  ANY-   8
           IS INPUT:      #OR
           IS INPUT:      #ELSE
           IS INPUT:      #THEN
           IS INPUT:      #END
           IS INPUT:   ")"
                          REDUCE TO BOOLEANTERM
                GO TO     DNT  19 .

158:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     302.
         TOP STACK SYMBOL:        BOOLEANTERM DUMMY
          SCAN;

         TOP STACK SYMBOL:      #AND
                          PUSH PAR  BOOLEANPRIMARY
          SCAN;  GO TO      TH  21 .

160:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     278.
```

```
                TOP STACK SYMBOL:       ARITHMETICEXPRESSI
                  IS INPUT:    "="
                 SCAN;

                TOP STACK SYMBOL:    "="
                                    PUSH BAR  ARITHMETICEXPRESSION
                 SCAN;  GO TO     TH  11 .

       162:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     281.
                TOP STACK SYMBOL:       ARITHMETICEXPRESSI
                  IS INPUT:    "≠"
                 SCAN;

                TOP STACK SYMBOL:    "≠"
                                    PUSH BAR  ARITHMETICEXPRESSION
                 SCAN;  GO TO     TH  11 .

       164:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     284.
                TOP STACK SYMBOL:       ARITHMETICEXPRESSI
                  IS INPUT:    ">"
                 SCAN;

                TOP STACK SYMBOL:    ">"
                                    PUSH BAR  ARITHMETICEXPRESSION
                 SCAN;  GO TO     TH  11 .

       166:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     287.
                TOP STACK SYMBOL:       ARITHMETICEXPRESSI
                  IS INPUT:    "<"
                 SCAN;

                TOP STACK SYMBOL:    "<"
                                    PUSH BAR  ARITHMETICEXPRESSION
                 SCAN;  GO TO     TH  11 .

       168:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     290.
                TOP STACK SYMBOL:       ARITHMETICEXPRESSI
                  IS INPUT:    "≤"
                 SCAN;

                TOP STACK SYMBOL:    "≤"
                                    PUSH BAR  ARITHMETICEXPRESSION
                 SCAN;  GO TO     TH  11 .

       170:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     293.
                TOP STACK SYMBOL:       ARITHMETICEXPRESSI
                 SCAN;

                TOP STACK SYMBOL:    "≥"
                                    PUSH BAR  ARITHMETICEXPRESSION
                 SCAN;  GO TO     TH  11 .

       172:     ERROR PRODUCTION:   SKIP SYMBOLS.

NTRN  44  GROUP    41;     BOOLEANEXPRESSION

       173:     PRODUCTION ASSOCIATED WITH PROTAB NO.:      45.
                TOP STACK SYMBOL:       BOOLEANEXPRESSION     @S-12
                                    REDUCE TO STATEMENT
                              GO TO    DNT    6 .
```

```
NTPN  48  GROUP   42:      BOOLEANEXPRESSION

          174:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     49.
                  TOP STACK SYMBOL:       BOOLEANEXPRESSION    BS-12
                                  REDUCE TO STATEMENT
                          GO TO   DNT   6 .

NTPN  89  GROUP   43:      BOOLEANEXPRESSION

          175:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     90.
                  TOP STACK SYMBOL:       BOOLEANEXPRESSION    BS-12
                                  REDUCE TO STATEMENT
                          GO TO   DNT   6 .

NTPN  94  GROUP   44:      BOOLEANEXPRESSION

          176:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     95.
                  TOP STACK SYMBOL:       BOOLEANEXPRESSION    BS-12
                                  REDUCE TO STATEMENT
                          GO TO   DNT   6 .
NTPN 276  GROUP   45:      BOOLEANEXPRESSION

          177:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    276.
                  TOP STACK SYMBOL:       BOOLEANEXPRESSION
                   SCAN:

                  TOP STACK SYMBOL:   ")"
                                  REDUCE TO BOOLEANPRIMARY
                          GO TO   DNT  21 .

  TH  13  GROUP   46:      ANYTHINGBUTSEMICOLON

          179:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    196.
                  TOP STACK SYMBOL:  <*I>
                                  REDUCE TO ANYTHINGBUTSEMICOLON
                          GO TO   DNT  13 .

          180:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    197.
                  TOP STACK SYMBOL:  <*N>
                                  REDUCE TO ANYTHINGBUTSEMICOLON
                          GO TO   DNT  13 .

          181:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    198.
                  TOP STACK SYMBOL:  <*S>
                                  REDUCE TO ANYTHINGBUTSEMICOLON
                          GO TO   DNT  13 .

          182:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    199.
                  TOP STACK SYMBOL:      #BEGIN
                                  REDUCE TO ANYTHINGBUTSEMICOLON
                          GO TO   DNT  13 .

          183:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    200.
                  TOP STACK SYMBOL:      #END
                                  REDUCE TO ANYTHINGBUTSEMICOLON
                          GO TO   DNT  13 .

          184:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    201.
                  TOP STACK SYMBOL:      #GO
                                  REDUCE TO ANYTHINGBUTSEMICOLON
                          GO TO   DNT  13 .
```

```
185:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     202.
         TOP STACK SYMBOL:      #TO
                         REDUCE TO ANYTHINGBUTSEMICOLON
                GO TO    DNT  13 .

186:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     203.
         TOP STACK SYMBOL:      #GOTO
                         REDUCE TO ANYTHINGBUTSEMICOLON
                GO TO    DNT  13 .

187:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     204.
         TOP STACK SYMBOL:      #COMMENT
                         REDUCE TO ANYTHINGBUTSEMICOLON
                GO TO    DNT  13 .

188:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     205.
         TOP STACK SYMBOL:      #INTEGER
                         REDUCE TO ANYTHINGBUTSEMICOLON
                GO TO    DNT  13 .

189:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     206.
         TOP STACK SYMBOL:      #BOOLEAN
                         REDUCE TO ANYTHINGBUTSEMICOLON
                GO TO    DNT  13 .

190:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     207.
         TOP STACK SYMBOL:      #LABEL
                         REDUCE TO ANYTHINGBUTSEMICOLON
                GO TO    DNT  13 .

191:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     208.
         TOP STACK SYMBOL:      #IF
                         REDUCE TO ANYTHINGBUTSEMICOLON
                GO TO    DNT  13 .

192:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     209.
         TOP STACK SYMBOL:      #THEN
                         REDUCE TO ANYTHINGBUTSEMICOLON
                GO TO    DNT  13 .

193:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     210.
         TOP STACK SYMBOL:      #ELSE
                         REDUCE TO ANYTHINGBUTSEMICOLON
                GO TO    DNT  13 .

194:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     211.
         TOP STACK SYMBOL:      #AND
                         REDUCE TO ANYTHINGBUTSEMICOLON
                GO TO    DNT  13 .

195:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     212.
         TOP STACK SYMBOL:      #OR
                         REDUCE TO ANYTHINGBUTSEMICOLON
                GO TO    DNT  13 .

196:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     213.
         TOP STACK SYMBOL:      #TRUE
                         REDUCE TO ANYTHINGBUTSEMICOLON
                GO TO    DNT  13 .
```

```
          197:     PRODUCTION ASSOCIATED WITH PROTAB NO.:    214.
                   TOP STACK SYMBOL:     #FALSE
                                       REDUCE TO ANYTHINGBUTSEMICOLON
                             GO TO    DNT  13 .

          198:     PRODUCTION ASSOCIATED WITH PROTAB NO.:    215.
                   TOP STACK SYMBOL:   ","
                                       REDUCE TO ANYTHINGBUTSEMICOLON
                             GO TO    DNT  13 .

          199:     PRODUCTION ASSOCIATED WITH PROTAB NO.:    216.
                   TOP STACK SYMBOL:    ":"
                                       REDUCE TO ANYTHINGBUTSEMICOLON
                             GO TO    DNT  13 .

          200:     PRODUCTION ASSOCIATED WITH PROTAB NO.:    217.
                   TOP STACK SYMBOL:    "#"
                                       REDUCE TO ANYTHINGBUTSEMICOLON
                             GO TO    DNT  13 .

          201:     PRODUCTION ASSOCIATED WITH PROTAB NO.:    218.
                   TOP STACK SYMBOL:    "@"
                                       REDUCE TO ANYTHINGBUTSEMICOLON
                             GO TO    DNT  13 .

          202:     PRODUCTION ASSOCIATED WITH PROTAB NO.:    219.
                   TOP STACK SYMBOL:    "."
                                       REDUCE TO ANYTHINGBUTSEMICOLON
                             GO TO    DNT  13 .

          203:     ERROR PRODUCTION:   MAKE REDUCTION.

   NTPN 221  GROUP    47:     ANYTHINGBUTSEMICOLON

          204:     PRODUCTION ASSOCIATED WITH PROTAB NO.:    221.
                   TOP STACK SYMBOL:      ANYTHINGBUTSEMICOL
                                       REDUCE TO COMMENT DUMMY 5
                             GO TO    DNT  14 .

   NTPN 223  GROUP    48:     ANYTHINGBUTSEMICOLON

          205:     PRODUCTION ASSOCIATED WITH PROTAB NO.:    223.
                   TOP STACK SYMBOL:      ANYTHINGBUTSEMICOL
                                       REDUCE TO COMMENT DUMMY 5
                             GO TO    DNT  14 .

     TH   15  GROUP    49:     TERM

              GROUP     TH  15  IS THE SAME AS GROUP     TH  11

    NTH   15  GROUP    49:     TERM

          206:     PRODUCTION ASSOCIATED WITH PROTAB NO.:    224.
                   TOP STACK SYMBOL:      ARITHMETICPRIMARY
                     IS INPUT:  ANY-   8
                     IS INPUT:       #AND
                     IS INPUT:       #OR
                     IS INPUT:       #ELSE
                     IS INPUT:       #THEN
                     IS INPUT:       #END
                     IS INPUT:    "="
```

```
               IS INPUT:    "≠"
               IS INPUT:    "≤"
               IS INPUT:    ")"
               IS INPUT:    "="
               IS INPUT:    "<"
               IS INPUT:    "+"
               IS INPUT:    "≥"
               IS INPUT:    ">"
                                REDUCE TO TERM
                    GO TO     DNT  15 .

207:        PRODUCTION ASSOCIATED WITH PROTAB NO.:    257.
            TOP STACK SYMBOL:       ARITHMETICPRIMARY
              IS INPUT:    "x"
             SCAN;

            TOP STACK SYMBOL:    "x"
                                PUSH PAR  ARITHMETICPRIMARY
             SCAN;  GO TO     TH  17 .

209:        PRODUCTION ASSOCIATED WITH PROTAB NO.:    261.
            TOP STACK SYMBOL:       ARITHMETICPRIMARY
             SCAN;

            TOP STACK SYMBOL:    "/"
                                PUSH PAR  ARITHMETICPRIMARY
             SCAN;  GO TO     TH  17 .

211:        PRODUCTION ASSOCIATED WITH PROTAB NO.:    225.
            TOP STACK SYMBOL:       TERM DUMMY 7
              IS INPUT:  ANY=   8
              IS INPUT:       #AND
              IS INPUT:       #OR
              IS INPUT:       #ELSE
              IS INPUT:       #THEN
              IS INPUT:       #END
              IS INPUT:    "="
              IS INPUT:    "≠"
              IS INPUT:    "≤"
              IS INPUT:    ")"
              IS INPUT:    "="
              IS INPUT:    "<"
              IS INPUT:    "+"
              IS INPUT:    "≥"
              IS INPUT:    ">"
                                REDUCE TO TERM
                    GO TO     DNT  15 .

212:        PRODUCTION ASSOCIATED WITH PROTAB NO.:    249.
            TOP STACK SYMBOL:       TERM DUMMY 7
              IS INPUT:    "x"
             SCAN;

            TOP STACK SYMBOL:    "x"
                                PUSH PAR  ARITHMETICPRIMARY
             SCAN;  GO TO     TH  17 .

214:        PRODUCTION ASSOCIATED WITH PROTAB NO.:    253.
            TOP STACK SYMBOL:       TERM DUMMY 7
             SCAN;
```

```
                        TOP STACK SYMBOL:    "/"
                                       PUSH BAR  ARITHMETICPRIMARY
                         SCAN;  GO TO     TH  17 .

             216:       ERROR PRODUCTION:    SKIP SYMBOLS.

 NTPN 228  GROUP        50:       TERM

             217:       PRODUCTION ASSOCIATED WITH PROTAB NO.:     229.
                        TOP STACK SYMBOL:        TERM      @S-16
                                       REDUCE TO ARITHMETICEXPRESSION DUMMY 6
                                  GO TO    DNT  16 .

 NTPN 232  GROUP        51:       TERM

             218:       PRODUCTION ASSOCIATED WITH PROTAB NO.:     233.
                        TOP STACK SYMBOL:        TERM      @S-16
                                       REDUCE TO ARITHMETICEXPRESSION DUMMY 6
                                  GO TO    DNT  16 .

 NTPN 236  GROUP        52:       TERM

             219:       PRODUCTION ASSOCIATED WITH PROTAB NO.:     237.
                        TOP STACK SYMBOL:        TERM      @S-16
                                       REDUCE TO ARITHMETICEXPRESSION DUMMY 6
                                  GO TO    DNT  16 .

 NTPN 240  GROUP        53:       TERM

             220:       PRODUCTION ASSOCIATED WITH PROTAB NO.:     241.
                        TOP STACK SYMBOL:        TERM      @S-16
                                       REDUCE TO ARITHMETICEXPRESSION DUMMY 6
                                  GO TO    DNT  16 .

   TH  17  GROUP        54:       ARITHMETICPRIMARY

           GROUP          TH  17  IS THE SAME AS GROUP      TH  11

 NTPN 251  GROUP        54:       ARITHMETICPRIMARY

             221:       PRODUCTION ASSOCIATED WITH PROTAB NO.:     252.
                        TOP STACK SYMBOL:        ARITHMETICPRIMARY    @S-16
                                       REDUCE TO TERM DUMMY 7
                                  GO TO    DNT  18 .

 NTPN 255  GROUP        55:       ARITHMETICPRIMARY

             222:       PRODUCTION ASSOCIATED WITH PROTAB NO.:     256.
                        TOP STACK SYMBOL:        ARITHMETICPRIMARY    @S-16
                                       REDUCE TO TERM DUMMY 7
                                  GO TO    DNT  18 .

 NTPN 259  GROUP        56:       ARITHMETICPRIMARY

             223:       PRODUCTION ASSOCIATED WITH PROTAB NO.:     260.
                        TOP STACK SYMBOL:        ARITHMETICPRIMARY    @S-16
                                       REDUCE TO TERM DUMMY 7
                                  GO TO    DNT  18 .

 NTPN 263  GROUP        57:       ARITHMETICPRIMARY
```

```
        224:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     264.
                 TOP STACK SYMBOL:      ARITHMETICPRIMARY    PS-16
                                  REDUCE TO TERM DUMMY 7
                           GO TO   DNT  18 .

 TH  19  GROUP   58:      BOOLEANTERM

         GROUP     TH  19  IS THE SAME AS GROUP     TH  12

NTH  19  GROUP   58:      BOOLEANTERM

        225:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     192.
                 TOP STACK SYMBOL:      TERM
                   IS INPUT:  ANY-   8
                   IS INPUT:      #AND
                   IS INPUT:      #OR
                   IS INPUT:      #ELSE
                   IS INPUT:      #THEN
                   IS INPUT:      #END
                   IS INPUT:    "="
                   IS INPUT:    "≠"
                   IS INPUT:    "≤"
                   IS INPUT:    ")"
                   IS INPUT:    "<"
                   IS INPUT:    "≥"
                   IS INPUT:    ">"
                                  REDUCE TO ARITHMETICEXPRESSION
                           GO TO   DNT  11 .

        226:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     234.
                 TOP STACK SYMBOL:      TERM
                   IS INPUT:    "+"
                  SCAN:

                 TOP STACK SYMBOL:    "+"
                                  PUSH PAR  TERM
                  SCAN:  GO TO    TH  15 .

        228:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     238.
                 TOP STACK SYMBOL:      TERM
                  SCAN:

                 TOP STACK SYMBOL:    "-"
                                  PUSH PAR  TERM
                  SCAN:  GO TO    TH  15 .

        230:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     193.
                 TOP STACK SYMBOL:      ARITHMETICEXPRESSI
                   IS INPUT:  ANY-   8
                   IS INPUT:      #AND
                   IS INPUT:      #OR
                   IS INPUT:      #ELSE
                   IS INPUT:      #THEN
                   IS INPUT:      #END
                   IS INPUT:    "="
                   IS INPUT:    "≠"
                   IS INPUT:    "≤"
                   IS INPUT:    ")"
                   IS INPUT:    "<"
                   IS INPUT:    "≥"
                   IS INPUT:    ">"
```

```
                    REDUCE TO ARITHMETICEXPRESSION
              GO TO   DNT  11 .

231:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     226.
          TOP STACK SYMBOL:       ARITHMETICEXPRESSI
            IS INPUT:   "+"
           SCAN;

          TOP STACK SYMBOL:    "+"
                          PUSH PAR  TERM
           SCAN;  GO TO    TH  15 .

233:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     230.
          TOP STACK SYMBOL:       ARITHMETICEXPRESSI
           SCAN;

          TOP STACK SYMBOL:    "-"
                          PUSH PAR  TERM
           SCAN;  GO TO    TH  15 .

235:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     224.
          TOP STACK SYMBOL:       ARITHMETICPRIMARY
            IS INPUT:  ANY-   8
            IS INPUT:      #AND
            IS INPUT:      #OR
            IS INPUT:      #ELSE
            IS INPUT:      #THEN
            IS INPUT:      #END
            IS INPUT:    "="
            IS INPUT:    "≠"
            IS INPUT:    "≤"
            IS INPUT:    ")"
            IS INPUT:    "-"
            IS INPUT:    "<"
            IS INPUT:    "+"
            IS INPUT:    "≥"
            IS INPUT:    ">"
                          REDUCE TO TERM
              GO TO   DNT  15 .

236:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     257.
          TOP STACK SYMBOL:       ARITHMETICPRIMARY
            IS INPUT:    "x"
           SCAN;

          TOP STACK SYMBOL:    "x"
                          PUSH PAR  ARITHMETICPRIMARY
           SCAN;  GO TO    TH  17 .

238:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     261.
          TOP STACK SYMBOL:       ARITHMETICPRIMARY
           SCAN;

          TOP STACK SYMBOL:    "/"
                          PUSH PAR  ARITHMETICPRIMARY
           SCAN;  GO TO    TH  17 .

240:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     225.
          TOP STACK SYMBOL:       TERM DUMMY 7
            IS INPUT:  ANY-   8
            IS INPUT:      #AND
```

```
          IS INPUT:      #OR
          IS INPUT:      #ELSE
          IS INPUT:      #THEN
          IS INPUT:      #END
          IS INPUT:    "="
          IS INPUT:    "≠"
          IS INPUT:    "≤"
          IS INPUT:    ")"
          IS INPUT:    "−"
          IS INPUT:    "<"
          IS INPUT:    "+"
          IS INPUT:    "≥"
          IS INPUT:    ">"
                         REDUCE TO TERM
                GO TO    DNT  15 .

241:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     249.
        TOP STACK SYMBOL:        TERM DUMMY 7
          IS INPUT:    "×"
         SCAN:

        TOP STACK SYMBOL:    "×"
                         PUSH PAR  ARITHMETICPRIMARY
         SCAN:  GO TO    TH  17 .

243:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     253.
        TOP STACK SYMBOL:        TERM DUMMY 7
         SCAN:

        TOP STACK SYMBOL:    "/"
                         PUSH PAR  ARITHMETICPRIMARY
         SCAN:  GO TO    TH  17 .

245:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     265.
        TOP STACK SYMBOL:        BOOLEANPRIMARY
          IS INPUT:  ANY-   8
          IS INPUT:      #OR
          IS INPUT:      #ELSE
          IS INPUT:      #THEN
          IS INPUT:      #END
          IS INPUT:    ")"
                         REDUCE TO BOOLEANTERM
                GO TO    DNT  19 .

246:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     306.
        TOP STACK SYMBOL:        BOOLEANPRIMARY
         SCAN:

        TOP STACK SYMBOL:      #AND
                         PUSH PAR  BOOLEANPRIMARY
         SCAN:  GO TO    TH  21 .

248:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     266.
        TOP STACK SYMBOL:        BOOLEANTERM DUMMY
          IS INPUT:  ANY-   8
          IS INPUT:      #OR
          IS INPUT:      #ELSE
          IS INPUT:      #THEN
          IS INPUT:      #END
          IS INPUT:    ")"
                         REDUCE TO BOOLEANTERM
```

```
              GO TO    DNT  19 .

249:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    302.
        TOP STACK SYMBOL:      BOOLEANTERM DUMMY
         SCAN;

        TOP STACK SYMBOL:     #AND
                       PUSH BAR  BOOLEANPRIMARY
         SCAN;  GO TO    TH  21 .

251:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    278.
        TOP STACK SYMBOL:      ARITHMETICEXPRESSI
          IS INPUT:   "="
         SCAN;

        TOP STACK SYMBOL:   "="
                       PUSH BAR  ARITHMETICEXPRESSION
         SCAN;  GO TO    TH  11 .

253:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    281.
        TOP STACK SYMBOL:      ARITHMETICEXPRESSI
          IS INPUT:   "≠"
         SCAN;

        TOP STACK SYMBOL:   "≠"
                       PUSH BAR  ARITHMETICEXPRESSION
         SCAN;  GO TO    TH  11 .

255:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    284.
        TOP STACK SYMBOL:      ARITHMETICEXPRESSI
          IS INPUT:   ">"
         SCAN;

        TOP STACK SYMBOL:   ">"
                       PUSH BAR  ARITHMETICEXPRESSION
         SCAN;  GO TO    TH  11 .

257:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    287.
        TOP STACK SYMBOL:      ARITHMETICEXPRESSI
          IS INPUT:   "<"
         SCAN;

        TOP STACK SYMBOL:   "<"
                       PUSH BAR  ARITHMETICEXPRESSION
         SCAN;  GO TO    TH  11 .

259:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    290.
        TOP STACK SYMBOL:      ARITHMETICEXPRESSI
          IS INPUT:   "≤"
         SCAN;

        TOP STACK SYMBOL:   "≤"
                       PUSH BAR  ARITHMETICEXPRESSION
         SCAN;  GO TO    TH  11 .

261:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    293.
        TOP STACK SYMBOL:      ARITHMETICEXPRESSI
         SCAN;

        TOP STACK SYMBOL:   ">"
                       PUSH BAR  ARITHMETICEXPRESSION
```

```
                   SCAN;  GO TO     TH  11 .

          263:     ERROR PRODUCTION:   SKIP SYMBOLS.

NTPN 269  GROUP    59:      BOOLEANTERM

          264:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     270.
                   TOP STACK SYMBOL:         BOOLEANTERM     @S-16
                                       REDUCE TO BOOLEANEXPRESSION DUMMY 8
                             GO TO     DNT  20 .

NTPN 273  GROUP    60:      BOOLEANTERM

          265:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     274.
                   TOP STACK SYMBOL:         BOOLEANTERM     @S-16
                                       REDUCE TO BOOLEANEXPRESSION DUMMY 8
                             GO TO     DNT  20 .

  TH  21  GROUP    61:      BOOLEANPRIMARY

          GROUP      TH  21  IS THE SAME AS GROUP     TH  12

 NTH  21  GROUP    61:      BOOLEANPRIMARY

          266:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     192.
                   TOP STACK SYMBOL:         TERM
                     IS INPUT:  ANY-    8
                     IS INPUT:      #AND
                     IS INPUT:      #OR
                     IS INPUT:      #ELSE
                     IS INPUT:      #THEN
                     IS INPUT:      #END
                     IS INPUT:    "="
                     IS INPUT:    "≠"
                     IS INPUT:    "≤"
                     IS INPUT:    ")"
                     IS INPUT:    "<"
                     IS INPUT:    "≥"
                     IS INPUT:    ">"
                                       REDUCE TO ARITHMETICEXPRESSION
                             GO TO     DNT  11 .

          267:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     234.
                   TOP STACK SYMBOL:         TERM
                     IS INPUT:    "+"
                    SCAN:

                   TOP STACK SYMBOL:    "+"
                                       PUSH PAR   TERM
                    SCAN:   GO TO     TH  15 .

          269:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     238.
                   TOP STACK SYMBOL:         TERM
                    SCAN:

                   TOP STACK SYMBOL:    "-"
                                       PUSH PAR   TERM
                    SCAN;   GO TO     TH  15 .

          271:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     193.
                   TOP STACK SYMBOL:         ARITHMETICEXPRESSI
```

```
            IS INPUT:  ANY=    8
            IS INPUT:      #AND
            IS INPUT:      #OR
            IS INPUT:      #ELSE
            IS INPUT:      #THEN
            IS INPUT:      #END
            IS INPUT:    "="
            IS INPUT:    "≠"
            IS INPUT:    "≤"
            IS INPUT:    ")"
            IS INPUT:    "<"
            IS INPUT:    "≥"
            IS INPUT:    ">"
                           REDUCE TO ARITHMETICEXPRESSION
                   GO TO   DNT  11 .

272:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     226.
          TOP STACK SYMBOL:        ARITHMETICEXPRESST
            IS INPUT:    "+"
           SCAN:

          TOP STACK SYMBOL:    "+"
                           PUSH PAR  TERM
           SCAN:  GO TO     TH  15 .

274:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     230.
          TOP STACK SYMBOL:        ARITHMETICEXPRESST
           SCAN:

          TOP STACK SYMBOL:    "-"
                           PUSH PAR  TERM
           SCAN:  GO TO     TH  15 .

276:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     224.
          TOP STACK SYMBOL:        ARITHMETICPRIMARY
            IS INPUT:  ANY=    8
            IS INPUT:      #AND
            IS INPUT:      #OR
            IS INPUT:      #ELSE
            IS INPUT:      #THEN
            IS INPUT:      #END
            IS INPUT:    "="
            IS INPUT:    "≠"
            IS INPUT:    "≤"
            IS INPUT:    ")"
            IS INPUT:    "-"
            IS INPUT:    "<"
            IS INPUT:    "+"
            IS INPUT:    "≥"
            IS INPUT:    ">"
                           REDUCE TO TERM
                   GO TO   DNT  15 .

277:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     257.
          TOP STACK SYMBOL:        ARITHMETICPRIMARY
            IS INPUT:    "x"
           SCAN:

          TOP STACK SYMBOL:    "x"
                           PUSH PAR  ARITHMETICPRIMARY
           SCAN:  GO TO     TH  17 .
```

```
279:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    261.
        TOP STACK SYMBOL:       ARITHMETICPRIMARY
         SCAN;

        TOP STACK SYMBOL:    "/"
                        PUSH PAR  ARITHMETICPRIMARY
         SCAN;  GO TO    TH  17 .

281:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    225.
        TOP STACK SYMBOL:       TERM DUMMY 7
          IS INPUT:  ANY=   8
          IS INPUT:      #AND
          IS INPUT:      #OR
          IS INPUT:      #ELSE
          IS INPUT:      #THEN
          IS INPUT:      #END
          IS INPUT:    "="
          IS INPUT:    "≠"
          IS INPUT:    "≤"
          IS INPUT:    ")"
          IS INPUT:    "-"
          IS INPUT:    "<"
          IS INPUT:    "+"
          IS INPUT:    "≥"
          IS INPUT:    ">"
                        REDUCE TO TERM
               GO TO    DNT  15 .

282:    PRODUCTION ASSOCIATED WITH PROTAB NO.:  . 249.
        TOP STACK SYMBOL:       TERM DUMMY 7
          IS INPUT:   "x"
         SCAN;

        TOP STACK SYMBOL:    "x"
                        PUSH PAR  ARITHMETICPRIMARY
         SCAN;  GO TO    TH  17 .

284:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    253.
        TOP STACK SYMBOL:       TERM DUMMY 7
         SCAN;

        TOP STACK SYMBOL:    "/"
                        PUSH PAR  ARITHMETICPRIMARY
         SCAN;  GO TO    TH  17 .

286:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    278.
        TOP STACK SYMBOL:       ARITHMETICEXPRESSI
          IS INPUT:   "="
         SCAN;

        TOP STACK SYMBOL:    "="
                        PUSH PAR  ARITHMETICEXPRESSION
         SCAN;  GO TO    TH  11 .

288:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    281.
        TOP STACK SYMBOL:       ARITHMETICEXPRESSI
          IS INPUT:   "≠"
         SCAN;

        TOP STACK SYMBOL:    "≠"
```

```
                                PUSH PAR   ARITHMETICEXPRFSSION
              SCAN;   GO TO      TH   11 .

      290:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     284.
               TOP STACK SYMBOL:          ARITHMETICEXPRFSSI
                 IS INPUT:     ">"
                SCAN;

               TOP STACK SYMBOL:     ">"
                                PUSH PAR   ARITHMETICEXPRFSSION
                SCAN;   GO TO      TH   11 .

      292:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     287.
               TOP STACK SYMBOL:          ARITHMFTICEXPRFSSI
                 IS INPUT:     "<"
                SCAN;

               TOP STACK SYMBOL:     "<"
                                PUSH PAR   ARITHMETICEXPRFSSION
                SCAN;   GO TO      TH   11 .

      294:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     290.
               TOP STACK SYMBOL:          ARITHMFTICEXPRFSSI
                 IS INPUT:     "≤"
                SCAN;

               TOP STACK SYMBOL:     "≤"
                                PUSH PAR   ARITHMETICEXPRESSION
                SCAN;   GO TO      TH   11 .

      296:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     293.
               TOP STACK SYMBOL:          ARITHMFTICEXPRFSSI
                SCAN;

               TOP STACK SYMBOL:     ">"
                                PUSH PAR   ARITHMETICEXPRESSION
                SCAN;   GO TO      TH   11 .

      298:     FRROR PRODUCTION:    SKIP SYMBOLS.

NTPN 304  GROUP   62:       BOOLFANPRIMARY

      299:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     305.
               TOP STACK SYMBOL:          BOOLEANPRIMARY          @S-16
                                REDUCF TO BOOLFANTFRM DUMMY 9
                        GO TO    DNT   22 .

NTPN 308  GROUP   63:       BOOLEANPRIMARY

      300:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     309.
               TOP STACK SYMBOL:          BOOLEANPRIMARY          @S-16
                                REDUCF TO BOOLFANTFRM DUMMY 9
                        GO TO    DNT   22 .

  TH   23  GROUP   64:       DUMMY 23

           GROUP     TH   23  IS THE SAME AS GROUP      TH   11

 NTH   23  GROUP   64:       DUMMY 23

      301:     PRODUCTION ASSOCIATED WITH PROTAB NO.:     192.
```

```
       TOP STACK SYMBOL:       TERM
         IS INPUT:  ANY=   8
         IS INPUT:      #AND
         IS INPUT:      #OR
         IS INPUT:      #ELSE
         IS INPUT:      #THEN
         IS INPUT:      #END
         IS INPUT:    "="
         IS INPUT:    "≠"
         IS INPUT:    "≤"
         IS INPUT:    ")"
         IS INPUT:    "<"
         IS INPUT:    "≥"
         IS INPUT:    ">"
                        REDUCE TO ARITHMETICEXPRESSION
                  GO TO   DNT  11 .

302:   PRODUCTION ASSOCIATED WITH PROTAB NO.:    234.
       TOP STACK SYMBOL:       TERM
         IS INPUT:    "+"
        SCAN;

       TOP STACK SYMBOL:    "+"
                        PUSH PAR  TERM
        SCAN;  GO TO     TH  15 .

304:   PRODUCTION ASSOCIATED WITH PROTAB NO.:    238.
       TOP STACK SYMBOL:       TERM
        SCAN;

       TOP STACK SYMBOL:    "-"
                        PUSH PAR  TERM
        SCAN;  GO TO     TH  15 .

306:   PRODUCTION ASSOCIATED WITH PROTAB NO.:    193.
       TOP STACK SYMBOL:       ARITHMETICEXPRESSI
         IS INPUT:  ANY=   8
         IS INPUT:      #AND
         IS INPUT:      #OR
         IS INPUT:      #ELSE
         IS INPUT:      #THEN
         IS INPUT:      #END
         IS INPUT:    "="
         IS INPUT:    "≠"
         IS INPUT:    "≤"
         IS INPUT:    ")"
         IS INPUT:    "<"
         IS INPUT:    "≥"
         IS INPUT:    ">"
                        REDUCE TO ARITHMETICEXPRESSION
                  GO TO   DNT  11 .

307:   PRODUCTION ASSOCIATED WITH PROTAB NO.:    226.
       TOP STACK SYMBOL:       ARITHMETICEXPRESSI
         IS INPUT:    "+"
        SCAN;

       TOP STACK SYMBOL:    "+"
                        PUSH PAR  TERM
        SCAN;  GO TO     TH  15 .
```

```
309:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     230.
        TOP STACK SYMBOL:       ARITHMETICEXPRESSI
         SCAN;

        TOP STACK SYMBOL:    "-"
                         PUSH BAR   TERM
         SCAN;  GO TO     TH  15 .

311:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     224.
        TOP STACK SYMBOL:       ARITHMETICPRIMARY
          IS INPUT:  ANY-    8
          IS INPUT:      #AND
          IS INPUT:      #OR
          IS INPUT:      #ELSE
          IS INPUT:      #THEN
          IS INPUT:      #END
          IS INPUT:    "="
          IS INPUT:    "≠"
          IS INPUT:    "≤"
          IS INPUT:    ")"
          IS INPUT:    "-"
          IS INPUT:    "<"
          IS INPUT:    "+"
          IS INPUT:    "≥"
          IS INPUT:    ">"
                         REDUCE TO TERM
                GO TO    DNT  15 .

312:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     257.
        TOP STACK SYMBOL:       ARITHMETICPRIMARY
          IS INPUT:    "x"
         SCAN;

        TOP STACK SYMBOL:    "x"
                         PUSH BAR   ARITHMETICPRIMARY
         SCAN;  GO TO     TH  17 .

314:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     261.
        TOP STACK SYMBOL:       ARITHMETICPRIMARY
         SCAN;

        TOP STACK SYMBOL:    "/"
                         PUSH BAR   ARITHMETICPRIMARY
         SCAN;  GO TO     TH  17 .

316:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     225.
        TOP STACK SYMBOL:       TERM DUMMY 7
          IS INPUT:  ANY-    8
          IS INPUT:      #AND
          IS INPUT:      #OR
          IS INPUT:      #ELSE
          IS INPUT:      #THEN
          IS INPUT:      #END
          IS INPUT:    "=".
          IS INPUT:    "≠"
          IS INPUT:    "≤"
          IS INPUT:    ")"
          IS INPUT:    "-"
          IS INPUT:    "<"
          IS INPUT:    "+"
          IS INPUT:    "≥"
```

```
              IS INPUT:   ">"
                              REDUCF TO TERM
                    GO TO     DNT  15 .

     317:       PRODUCTION ASSOCIATED WITH PROTAB NO.:     249.
                TOP STACK SYMBOL:        TERM DUMMY 7
                  IS INPUT:   "x"
                 SCAN:

                TOP STACK SYMBOL:    "x"
                              PUSH BAR  ARITHMETICPRIMARY
                 SCAN:  GO TO     TH  17 .

     319:       PRODUCTION ASSOCIATED WITH PROTAB NO.:     253.
                TOP STACK SYMBOL:        TERM DUMMY 7
                 SCAN:

                TOP STACK SYMBOL:    "/"
                              PUSH BAR  ARITHMETICPRIMARY
                 SCAN:  GO TO     TH  17 .

     321:       PRODUCTION ASSOCIATED WITH PROTAB NO.:     310.
                TOP STACK SYMBOL:        ARITHMFTICEXPRESST
                 SCAN:

                TOP STACK SYMBOL:    ")"
                              REDUCF TO DUMMY 23
                    GO TO     DNT  23 .

     323:       ERROR PRODUCTION:   SKIP SYMBOLS.

NTPN 243  GROUP    65:      DUMMY 23

     324:       PRODUCTION ASSOCIATED WITH PROTAB NO.:     243.
                TOP STACK SYMBOL:        DUMMY 23
                              REDUCF TO ARITHMETICPRIMARY
                    GO TO     DNT  17 .

CNTPN    0  GROUP    66:

    SOURCE OF THIS COMBINATION IS:    NTH     1

     325:       PRODUCTION ASSOCIATED WITH PROTAB NO.:     157.
                TOP STACK SYMBOL:        COMMENT
                  IS INPUT:  ANY-    7
                              PUSH BAR   STATEMENT
                 SCAN:  GO TO     TH   6 .

     326:       PRODUCTION ASSOCIATED WITH PROTAB NO.:     165.
                TOP STACK SYMBOL:        COMMENT
                              REDUCF TO BLOCK DUMMY 2
                    GO TO     DNT   7 .

CNTPN    1  GROUP    67:      BOOLEANEXPRESSION

    SOURCE OF THIS COMBINATION IS:     TH     6

     327:       COMBINED PRODUCTION ASSOCIATED WITH PROTAB NO.:
                          52,  109,  117,  143,
                TOP STACK SYMBOL:        BOOLEANEXPRESSION      @S-13
                 SCAN:  GO TO  CTPN    6 .
```

```
CTPN   2  GROUP   68:

    SOURCE OF THIS COMBINATION IS:     NTH    6

        328:      PRODUCTION ASSOCIATED WITH PROTAB NO.:      60.
                  TOP STACK SYMBOL:        #GO
                    IS INPUT:      #TO
                   SCAN;

                  TOP STACK SYMBOL:        #TO
                   SCAN;

                  TOP STACK SYMBOL:  <*I>    @S=9
                                        REDUCE TO STATEMENT
                          GO TO    DNT   6 .

        331:      PRODUCTION ASSOCIATED WITH PROTAB NO.:      65.
                  TOP STACK SYMBOL:        #GO
                   SCAN;

                  TOP STACK SYMBOL:  <*I>    @S=9
                                        REDUCE TO STATEMENT
                          GO TO    DNT   6 .

CTPN   3  GROUP   69:

    SOURCE OF THIS COMBINATION IS:     NTH    6

        333:      PRODUCTION ASSOCIATED WITH PROTAB NO.:      74.
                  TOP STACK SYMBOL:  <*I>    @T=10
                    IS INPUT:    ":"
                   SCAN;

                  TOP STACK SYMBOL:    ":"
                   SCAN;

                  TOP STACK SYMBOL:    "="
                                        PUSH PAR  ARITHMETICEXPRESSION
                   SCAN;  GO TO     TH  11 .

        336:      PRODUCTION ASSOCIATED WITH PROTAB NO.:      81.
                  TOP STACK SYMBOL:  <*I>    @T=10
                   SCAN;

                  TOP STACK SYMBOL:    "+"
                                        PUSH PAR  ARITHMETICEXPRESSION
                   SCAN;  GO TO     TH  11 .

        338:      PRODUCTION ASSOCIATED WITH PROTAB NO.:      86.
                  TOP STACK SYMBOL:  <*I>
                    IS INPUT:    ":"    "="
                   SCAN;

                  TOP STACK SYMBOL:    ":"
                   SCAN;

                  TOP STACK SYMBOL:    "="
                                        PUSH PAR  BOOLEANEXPRESSION
                   SCAN;  GO TO     TH  12 .
```

```
        341:    PRODUCTION ASSOCIATED WITH PROTAB NO.:      92.
                TOP STACK SYMBOL:  <*T>
                  IS INPUT:   "+"
                 SCAN;

                TOP STACK SYMBOL:   "+"
                                PUSH BAR  BOOLEANEXPRESSION
                 SCAN;  GO TO    TH  12 .

        343:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     187.
                TOP STACK SYMBOL:  <*T>   @S-8
                 SCAN;

                TOP STACK SYMBOL:   ";"
                                REDUCE TO STATEMENT DUMMY 4
                      GO TO   DNT  10 .

CTPN   4  GROUP   70:

    SOURCE OF THIS COMBINATION IS:    NTH    6

        345:    COMBINED PRODUCTION ASSOCIATED WITH PROTAB NO.:
                         97,  124,  133,  149,
                TOP STACK SYMBOL:       #IF
                                PUSH SBR  BOOLEANEXPRESSION
                 SCAN;  GO TO    TH  12 .

 CTH   5  GROUP   71:

    SOURCE OF THIS COMBINATION IS:     TH    12

        GROUP    CTH   5  IS THE SAME AS GROUP     TH  12

CNTH   5  GROUP   71:     TERM

    SOURCE OF THIS COMBINATION IS:     TH    12

        346:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     192.
                TOP STACK SYMBOL:        TERM
                  IS INPUT:  ANY-   8
                  IS INPUT:      #AND
                  IS INPUT:      #OR
                  IS INPUT:      #ELSE
                  IS INPUT:      #THEN
                  IS INPUT:      #END
                  IS INPUT:    "="
                  IS INPUT:    "≠"
                  IS INPUT:    "≤"
                  IS INPUT:    ")"
                  IS INPUT:    "<"
                  IS INPUT:    "≥"
                  IS INPUT:    ">"
                                REDUCE TO ARITHMETICEXPRESSION
                      GO TO   DNT  11 .

        347:    PRODUCTION ASSOCIATED WITH PROTAB NO.:     234.
                TOP STACK SYMBOL:        TERM
                  IS INPUT:   "+"
                 SCAN;

                TOP STACK SYMBOL:   "+"
```

```
                      PUSH BAR  TERM
      SCAN;  GO TO     TH  15 .

349:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    238.
        TOP STACK SYMBOL:       TERM
         SCAN;

        TOP STACK SYMBOL:    "-"
                      PUSH BAR  TERM
         SCAN;  GO TO     TH  15 .

351:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    193.
        TOP STACK SYMBOL:       ARITHMETICEXPRESSI
          IS INPUT:  ANY=   8
          IS INPUT:      #AND
          IS INPUT:      #OR
          IS INPUT:      #ELSE
          IS INPUT:      #THEN
          IS INPUT:      #END
          IS INPUT:    "="
          IS INPUT:    "≠"
          IS INPUT:    "≤"
          IS INPUT:    ")"
          IS INPUT:    "<"
          IS INPUT:    "≥"
          IS INPUT:    ">"
                      REDUCE TO ARITHMETICEXPRESSION
              GO TO   DNT  11 .

352:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    226.
        TOP STACK SYMBOL:       ARITHMETICEXPRESSI
          IS INPUT:    "+"
         SCAN:

        TOP STACK SYMBOL:    "+"
                      PUSH BAR  TERM
         SCAN;  GO TO     TH  15 .

354:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    230.
        TOP STACK SYMBOL:       ARITHMETICEXPRESSI
         SCAN;

        TOP STACK SYMBOL:    "-"
                      PUSH BAR  TERM
         SCAN;  GO TO     TH  15 .

356:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    194.
        TOP STACK SYMBOL:       BOOLEANTERM
          IS INPUT:  ANY=   8
          IS INPUT:      #ELSE
          IS INPUT:      #THEN
          IS INPUT:      #END
          IS INPUT:    ")"
                      REDUCE TO BOOLEANEXPRESSION
              GO TO   DNT  12 .

357:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    271.
        TOP STACK SYMBOL:       BOOLEANTERM
         SCAN;

        TOP STACK SYMBOL:      #OR
```

```
                         PUSH BAR  BOOLEANTERM
      SCAN;  GO TO        TH  19 .

359:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     195.
          TOP STACK SYMBOL:       BOOLEANEXPRESSION
            IS INPUT:  ANY=   8
            IS INPUT:      #ELSE
            IS INPUT:      #THEN
            IS INPUT:      #END
            IS INPUT:    ")"
                         REDUCE TO BOOLEANEXPRESSION
                GO TO    DNT  12 .

360:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     267.
          TOP STACK SYMBOL:       BOOLEANEXPRESSION
           SCAN;

          TOP STACK SYMBOL:      #OR
                         PUSH BAR  BOOLEANTERM
           SCAN;  GO TO   TH  19 .

362:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     224.
          TOP STACK SYMBOL:       ARITHMETICPRIMARY
            IS INPUT:  ANY=   8
            IS INPUT:      #AND
            IS INPUT:      #OR
            IS INPUT:      #ELSE
            IS INPUT:      #THEN
            IS INPUT:      #END
            IS INPUT:    "="
            IS INPUT:    "≠"
            IS INPUT:    "≤"
            IS INPUT:    ")"
            IS INPUT:    "-"
            IS INPUT:    "<"
            IS INPUT:    "+"
            IS INPUT:    "≥"
            IS INPUT:    ">"
                         REDUCE TO TERM
                GO TO    DNT  15 .

363:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     257.
          TOP STACK SYMBOL:       ARITHMETICPRIMARY
            IS INPUT:    "x"
           SCAN;

          TOP STACK SYMBOL:    "x"
                         PUSH BAR  ARITHMETICPRIMARY
           SCAN;  GO TO   TH  17 .

365:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     261.
          TOP STACK SYMBOL:       ARITHMETICPRIMARY
           SCAN;

          TOP STACK SYMBOL:    "/"
                         PUSH BAR  ARITHMETICPRIMARY
           SCAN;  GO TO   TH  17 .

367:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     225.
          TOP STACK SYMBOL:       TERM DUMMY 7
            IS INPUT:  ANY=   8
```

```
      IS INPUT:     #AND
      IS INPUT:     #OR
      IS INPUT:     #ELSE
      IS INPUT:     #THEN
      IS INPUT:     #END
      IS INPUT:   "="
      IS INPUT:   "≠"
      IS INPUT:   "≤"
      IS INPUT:   ")"
      IS INPUT:   "="
      IS INPUT:   "<"
      IS INPUT:   "+"
      IS INPUT:   "≥"
      IS INPUT:   ">"
                    REDUCE TO TERM
             GO TO  DNT  15 .

368:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    249.
        TOP STACK SYMBOL:      TERM DUMMY 7
          IS INPUT:   "x"
         SCAN:

        TOP STACK SYMBOL:    "x"
                    PUSH PAR  ARITHMETICPRIMARY
         SCAN:  GO TO    TH  17 .

370:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    253.
        TOP STACK SYMBOL:      TERM DUMMY 7
         SCAN:

        TOP STACK SYMBOL:    "/"
                    PUSH PAR  ARITHMETICPRIMARY
         SCAN:  GO TO    TH  17 .

372:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    265.
        TOP STACK SYMBOL:      BOOLEANPRIMARY
          IS INPUT:  ANY=   8
          IS INPUT:     #OR
          IS INPUT:     #ELSE
          IS INPUT:     #THEN
          IS INPUT:     #END
          IS INPUT:   ")"
                    REDUCE TO BOOLEANTERM
             GO TO  DNT  19 .

373:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    306.
        TOP STACK SYMBOL:      BOOLEANPRIMARY
         SCAN:

        TOP STACK SYMBOL:      #AND
                    PUSH PAR  BOOLEANPRIMARY
         SCAN:  GO TO    TH  21 .

375:    PRODUCTION ASSOCIATED WITH PROTAB NO.:    266.
        TOP STACK SYMBOL:      BOOLEANTERM DUMMY
          IS INPUT:  ANY=   8
          IS INPUT:     #OR
          IS INPUT:     #ELSE
          IS INPUT:     #THEN
          IS INPUT:     #END
          IS INPUT:   ")"
```

```
                    REDUCE TO BOOLEANTERM
          GO TO     DNT  19 .

376:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     302.
          TOP STACK SYMBOL:       BOOLEANTERM DUMMY
           SCAN;

          TOP STACK SYMBOL:      #AND
                          PUSH BAR  BOOLEANPRIMARY
           SCAN;  GO TO    TH  21 .

378:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     278.
          TOP STACK SYMBOL:       ARITHMETICEXPRESSI
            IS INPUT:   "="
           SCAN;

          TOP STACK SYMBOL:    "="
                          PUSH BAR  ARITHMETICEXPRESSION
           SCAN;  GO TO    TH  11 .

380:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     281.
          TOP STACK SYMBOL:       ARITHMETICEXPRESSI
            IS INPUT:   "≠"
           SCAN;

          TOP STACK SYMBOL:    "≠"
                          PUSH BAR  ARITHMETICEXPRESSION
           SCAN;  GO TO    TH  11 .

382:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     284.
          TOP STACK SYMBOL:       ARITHMETICEXPRESSI
            IS INPUT:   ">"
           SCAN;

          TOP STACK SYMBOL:    ">"
                          PUSH BAR  ARITHMETICEXPRESSION
           SCAN;  GO TO    TH  11 .

384:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     287.
          TOP STACK SYMBOL:       ARITHMETICEXPRESSI
            IS INPUT:   "<"
           SCAN;

          TOP STACK SYMBOL:    "<"
                          PUSH BAR  ARITHMETICEXPRESSION
           SCAN;  GO TO    TH  11 .

386:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     290.
          TOP STACK SYMBOL:       ARITHMETICEXPRESSI
            IS INPUT:   "≤"
           SCAN;

          TOP STACK SYMBOL:    "≤"
                          PUSH BAR  ARITHMETICEXPRESSION
           SCAN;  GO TO    TH  11 .

388:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     293.
          TOP STACK SYMBOL:       ARITHMETICEXPRESSI
            IS INPUT:   "≥"
           SCAN;
```

```
                    TOP STACK SYMBOL:    ">"
                                        PUSH BAR  ARITHMETICEXPRESSION
                     SCAN;  GO TO      TH  11 .

          390:      PRODUCTION ASSOCIATED WITH PROTAB NO.:      310.
                    TOP STACK SYMBOL:         ARITHMETICEXPRESSI
                     SCAN;

                    TOP STACK SYMBOL:    ")"
                                        REDUCE TO DUMMY 23
                          GO TO      DNT  23 .

          392:      ERROR PRODUCTION:    SKIP SYMBOLS.

CTPN   6  GROUP   72:

    SOURCE OF THIS COMBINATION IS:  CNTPN     1

          393:      COMBINED PRODUCTION ASSOCIATED WITH PROTAB NO.:
                                53,  118,
                    TOP STACK SYMBOL:         #THEN
                     IS INPUT:  ANY-    7
                                        PUSH SBR  STATEMENT
                     SCAN:  GO TO      TH   6 .

          394:      COMBINED PRODUCTION ASSOCIATED WITH PROTAB NO.:
                                111,  145,
                    TOP STACK SYMBOL:         #THEN        @S-14
                     SCAN;  GO TO  CTPN    9 .

CNTPN   7  GROUP   73:       BOOLEANEXPRESSION

    SOURCE OF THIS COMBINATION IS:   CTPN     4

          395:      COMBINED PRODUCTION ASSOCIATED WITH PROTAB NO.:
                                99,  126,  135,  151,
                    TOP STACK SYMBOL:         BOOLEANEXPRESSION    @S-13
                     SCAN;  GO TO  CTPN  10 .

CNTPN   8  GROUP   74:       STATEMENT

    SOURCE OF THIS COMBINATION IS:   CTPN     6

          396:      COMBINED PRODUCTION ASSOCIATED WITH PROTAB NO.:
                                55,  120,
                    TOP STACK SYMBOL:         STATEMENT        @S-14
                     SCAN;  GO TO  CTPN  11 .

CTPN   9  GROUP   75:

    SOURCE OF THIS COMBINATION IS:   CTPN     6

          397:      PRODUCTION ASSOCIATED WITH PROTAB NO.:      112.
                    TOP STACK SYMBOL:         #ELSE
                     IS INPUT:  ANY-    7
                                        PUSH BAR  STATEMENT
                     SCAN:  GO TO      TH   6 .

          398:      PRODUCTION ASSOCIATED WITH PROTAB NO.:      147.
                    TOP STACK SYMBOL:         #ELSE      @S-15
                                        REDUCE TO STATEMENT
```

```
                    GO TO   DNT   6 .

CTPN  10  GROUP   76:

   SOURCE OF THIS COMBINATION IS:  CNTPN    7

      399:      COMBINED PRODUCTION ASSOCIATED WITH PROTAB NO.:
                         100,  136,
                TOP STACK SYMBOL:      #THEN
                  IS INPUT:  ANY=   7
                                PUSH SBR  STATEMENT
                 SCAN;  GO TO    TH   6 .

      400:      COMBINED PRODUCTION ASSOCIATED WITH PROTAB NO.:
                         128,  153,
                TOP STACK SYMBOL:      #THEN      @S=14
                 SCAN;  GO TO  CTPN  13 .

CTPN  11  GROUP   77:

   SOURCE OF THIS COMBINATION IS:  CNTPN    8

      401:      PRODUCTION ASSOCIATED WITH PROTAB NO.:      56.
                TOP STACK SYMBOL:      #ELSE
                  IS INPUT:  ANY=   7
                                PUSH BAR  STATEMENT
                 SCAN;  GO TO    TH   6 .

      402:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     122.
                TOP STACK SYMBOL:      #ELSE      @S=15
                                REDUCE TO STATEMENT
                        GO TO   DNT   6 .

CNTPN  12  GROUP   78:      STATEMENT

   SOURCE OF THIS COMBINATION IS:   CTPN    10

      403:      COMBINED PRODUCTION ASSOCIATED WITH PROTAB NO.:
                         102,  138,
                TOP STACK SYMBOL:       STATEMENT      @S=14
                 SCAN;  GO TO  CTPN  14 .

CTPN  13  GROUP   79:

   SOURCE OF THIS COMBINATION IS:   CTPN    10

      404:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     129.
                TOP STACK SYMBOL:      #ELSE
                  IS INPUT:  ANY=   7
                                PUSH BAR  STATEMENT
                 SCAN:  GO TO    TH   6 .

      405:      PRODUCTION ASSOCIATED WITH PROTAB NO.:     155.
                TOP STACK SYMBOL:      #ELSE      @S=15
                                REDUCE TO STATEMENT
                        GO TO   DNT   6 .

CTPN  14  GROUP   80:

   SOURCE OF THIS COMBINATION IS:  CNTPN    12
```

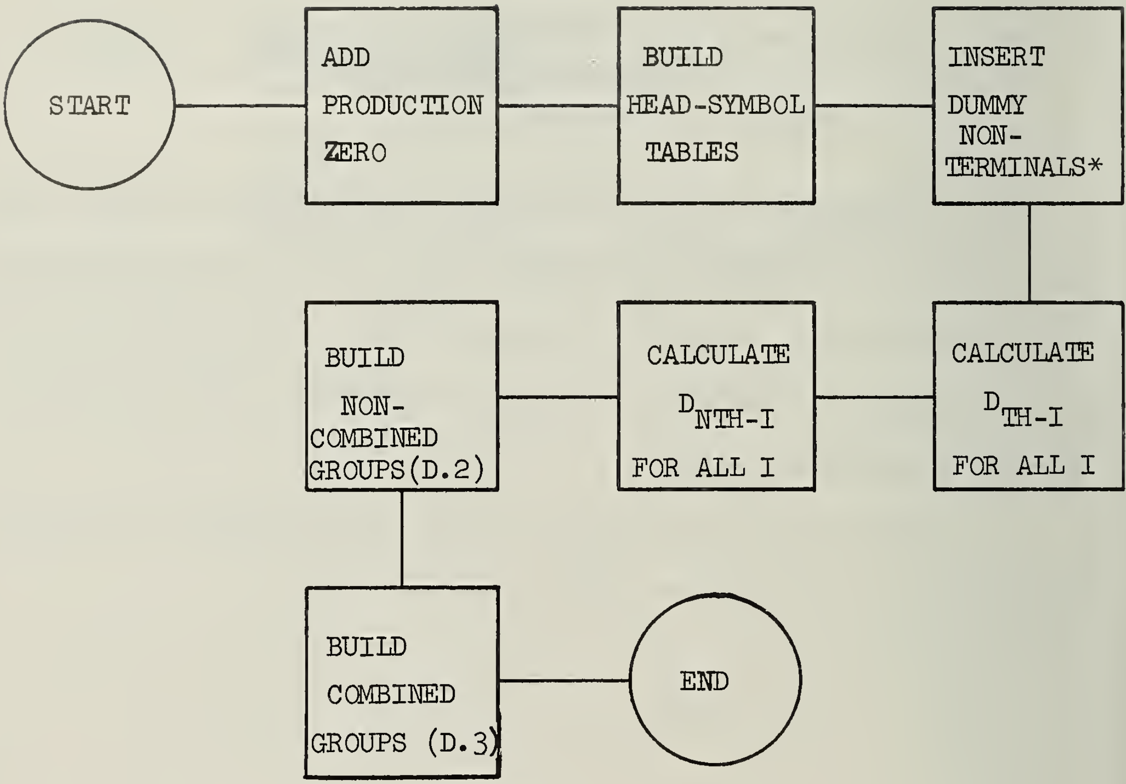
```
406:     PRODUCTION ASSOCIATED WITH PROTAB NO.:    103.
         TOP STACK SYMBOL:       #ELSE
           IS INPUT:  ANY=   7
                            PUSH BAR  STATEMENT
          SCAN;  GO TO    TH   6 .

407:     PRODUCTION ASSOCIATED WITH PROTAB NO.:    140.
         TOP STACK SYMBOL:       #ELSE     @S=15
                            REDUCE TO STATEMENT
                 GO TO    DNT   6 .
```

APPENDIX D

IMPLEMENTATION FLOW CHARTS

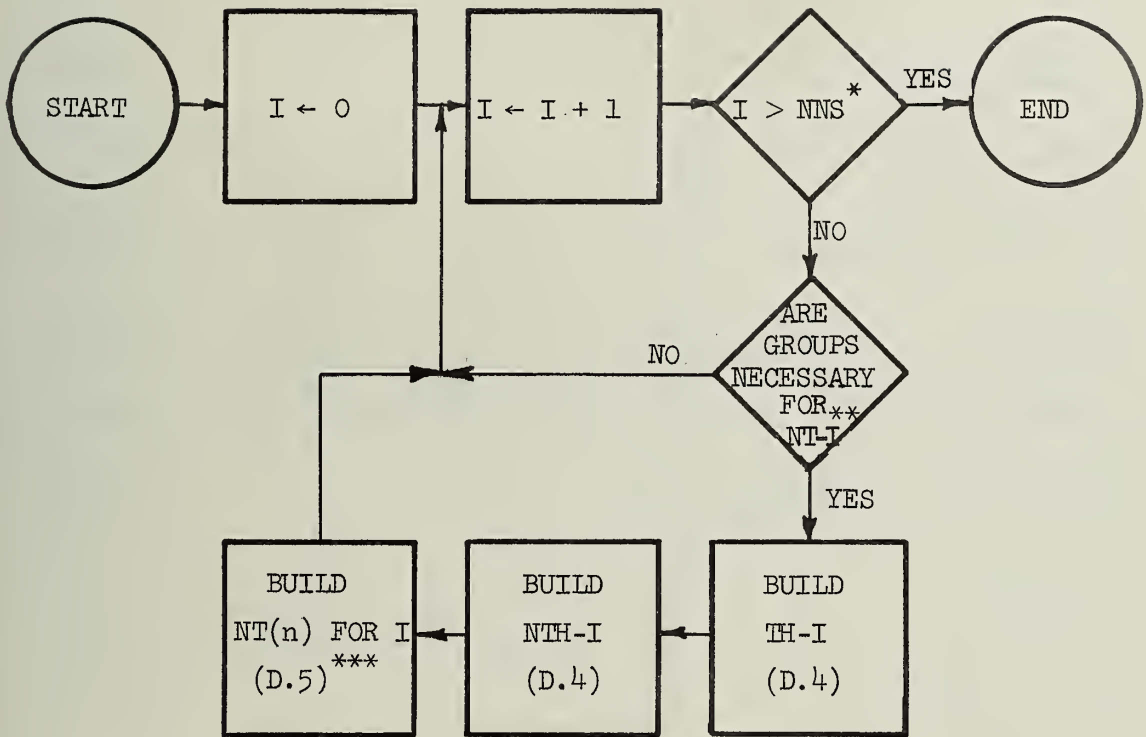
These flow charts, Figures D.1 through D.10, show the basic structure of the implementation of the CF to FPL conversion algorithm. They are relatively complete. The only things of any importance which are omitted, are those which are highly dependent on the methods of storing intermediate data.

Any box which includes a reference of the form (D.n) can be replaced by the entire flow chart of Figure D.n.



*Update head-symbol tables as dummies are inserted

D.1 Outer Structure

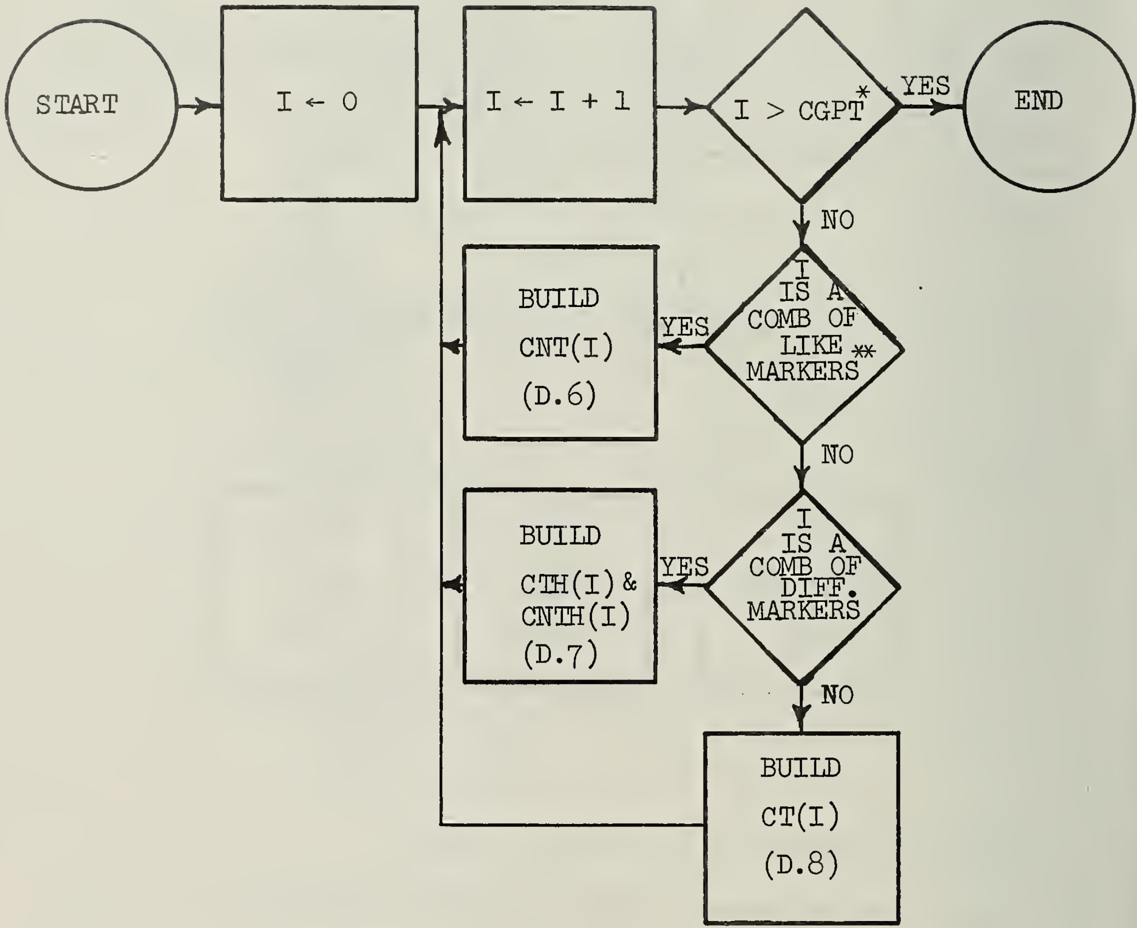


* NNS is the number of nonterminal symbols

** groups are necessary for NT-I if NT-I appears as a nonfirst symbol on some RHS

*** NT(n) is built for each nonfirst occurrence of I at n

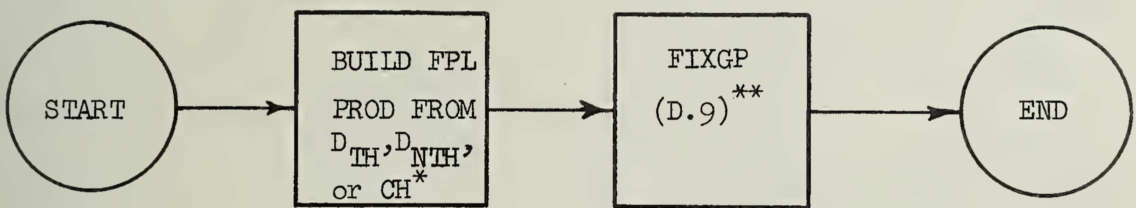
D.2 Construction of Noncombined Groups



* CGPT points to last entry in table of combined groups

** for the same nonterminal symbol

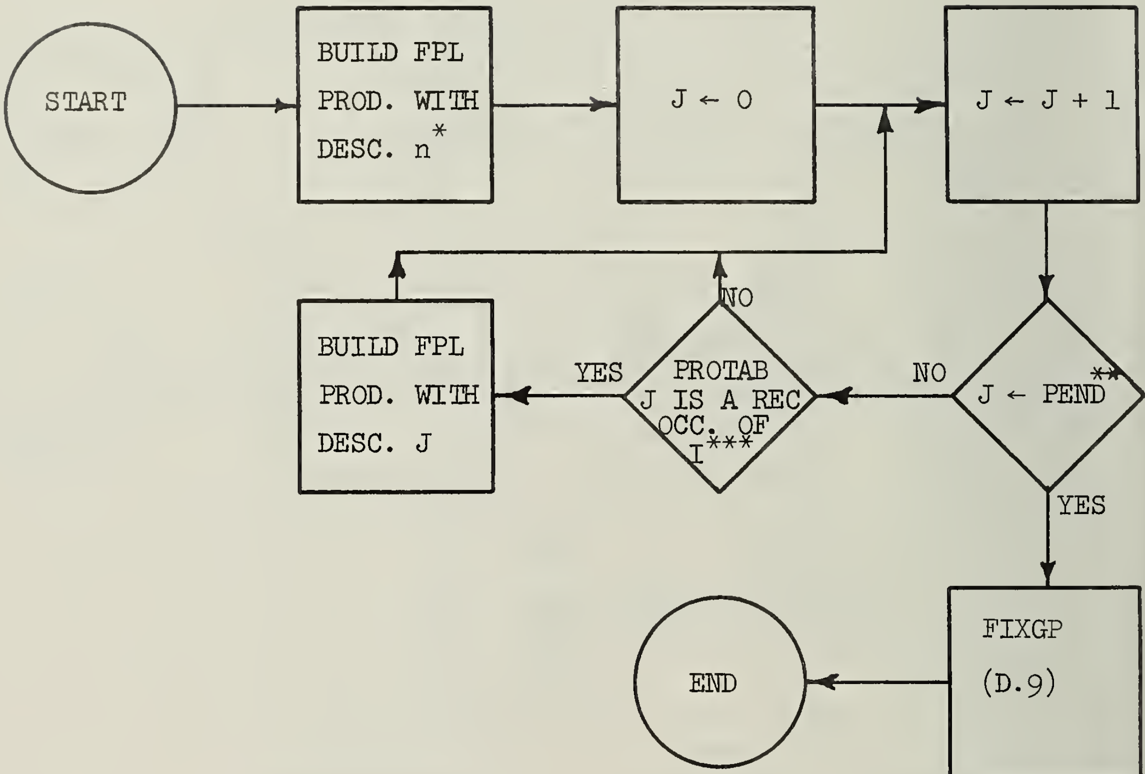
D.3 Construction of Combined Groups



* use mapping rules of section 3.3 on appropriate descriptor set

** this uses appropriate lookahead and/or combination to eliminate preclusions and generates appropriate T (π, n) productions inline

D.4 Construction of TH, NTH, CTH and CNTH Groups from Descriptor Sets

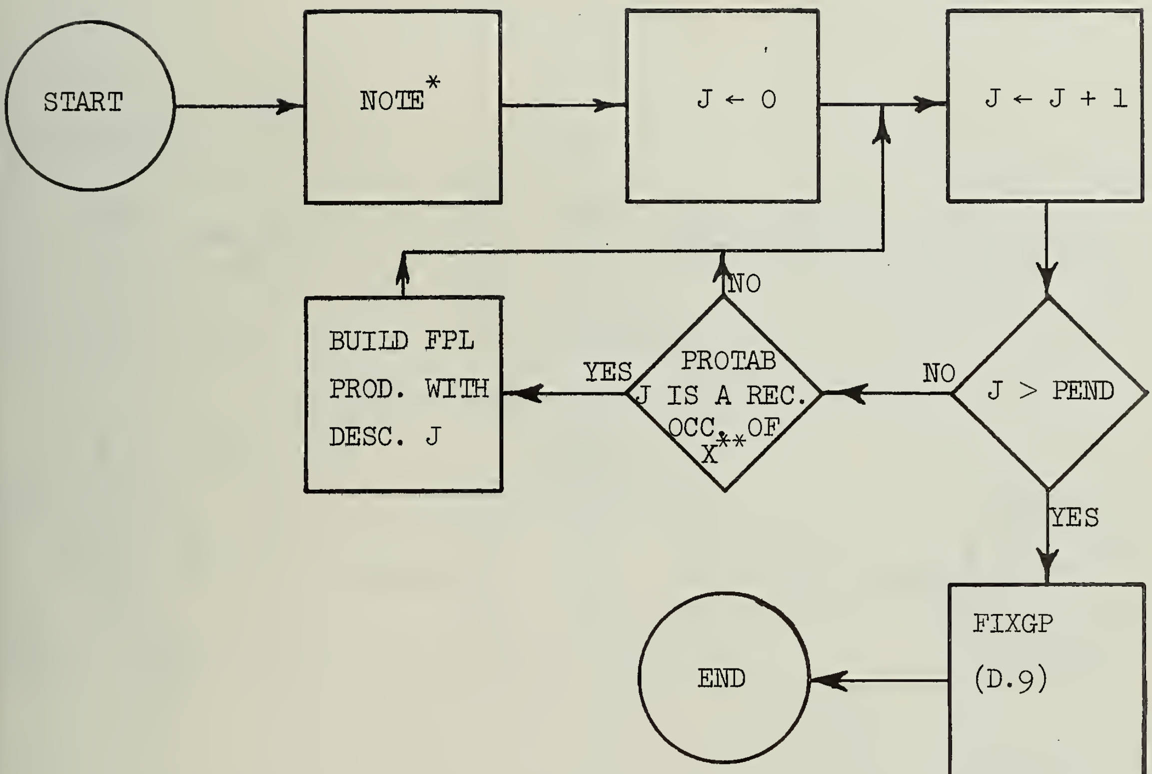


*   see D.2

**   PEND points to last entry in PRØTAB

***   see D.2

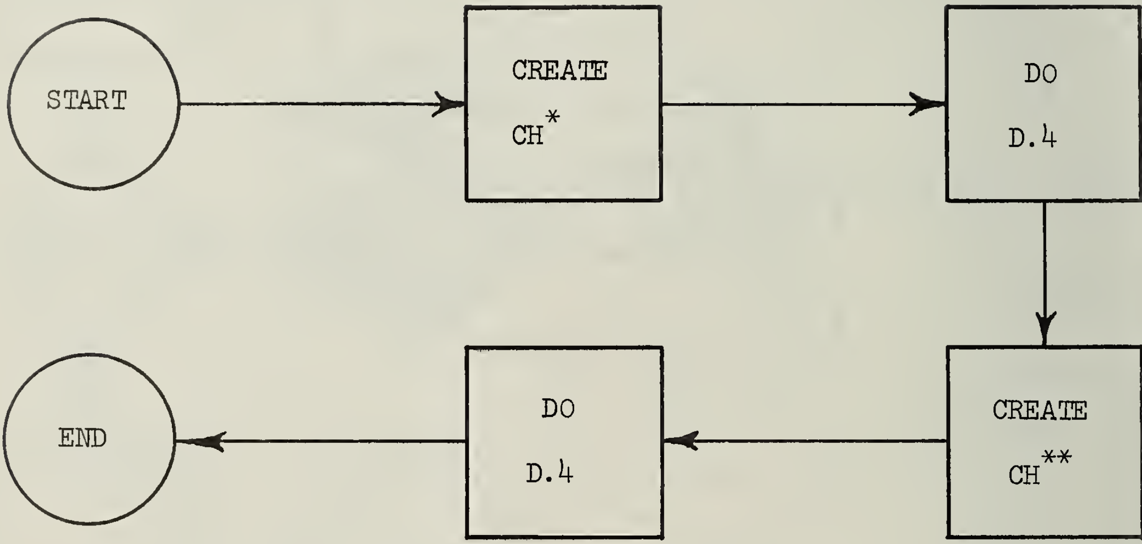
D.5 Construction of NT (n) Groups for Nonterminal I



* Build FPL productions with descriptors attached to the markers in combined group I (see D.3)

** X is the nonterminal symbol whose markers are in combined group I

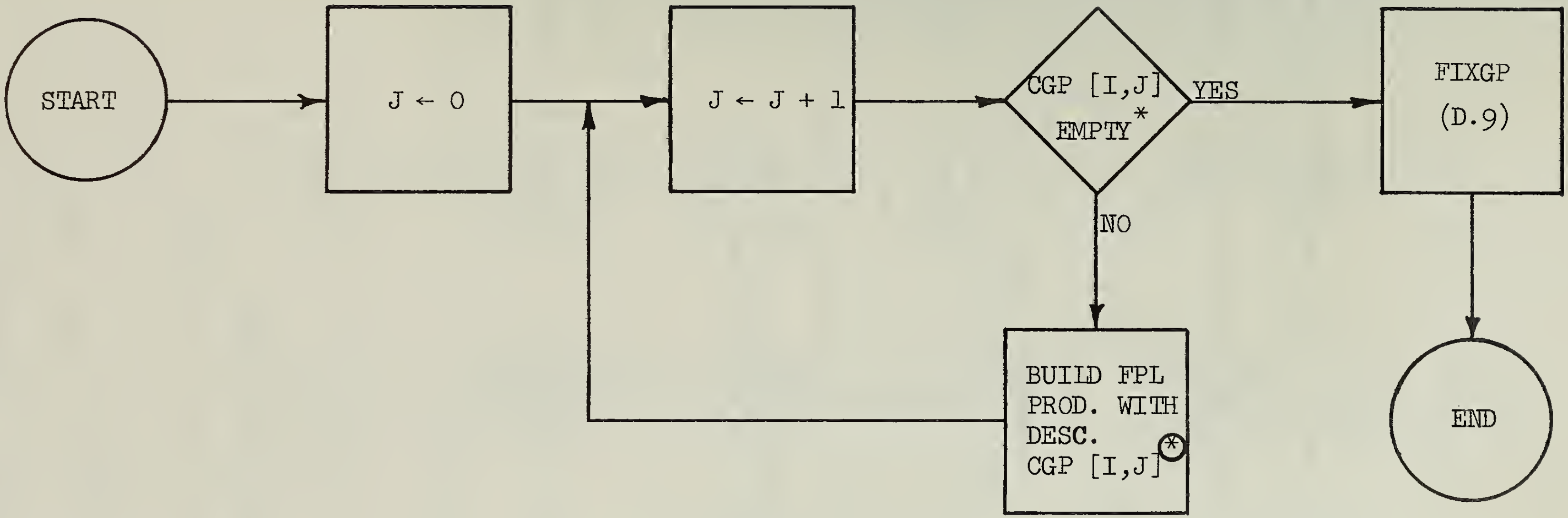
D.6 Construction of CNT-I Groups



* CH is the union of D_{TH-X} for each nonterminal X which has a marker in combined group I (see D.3)

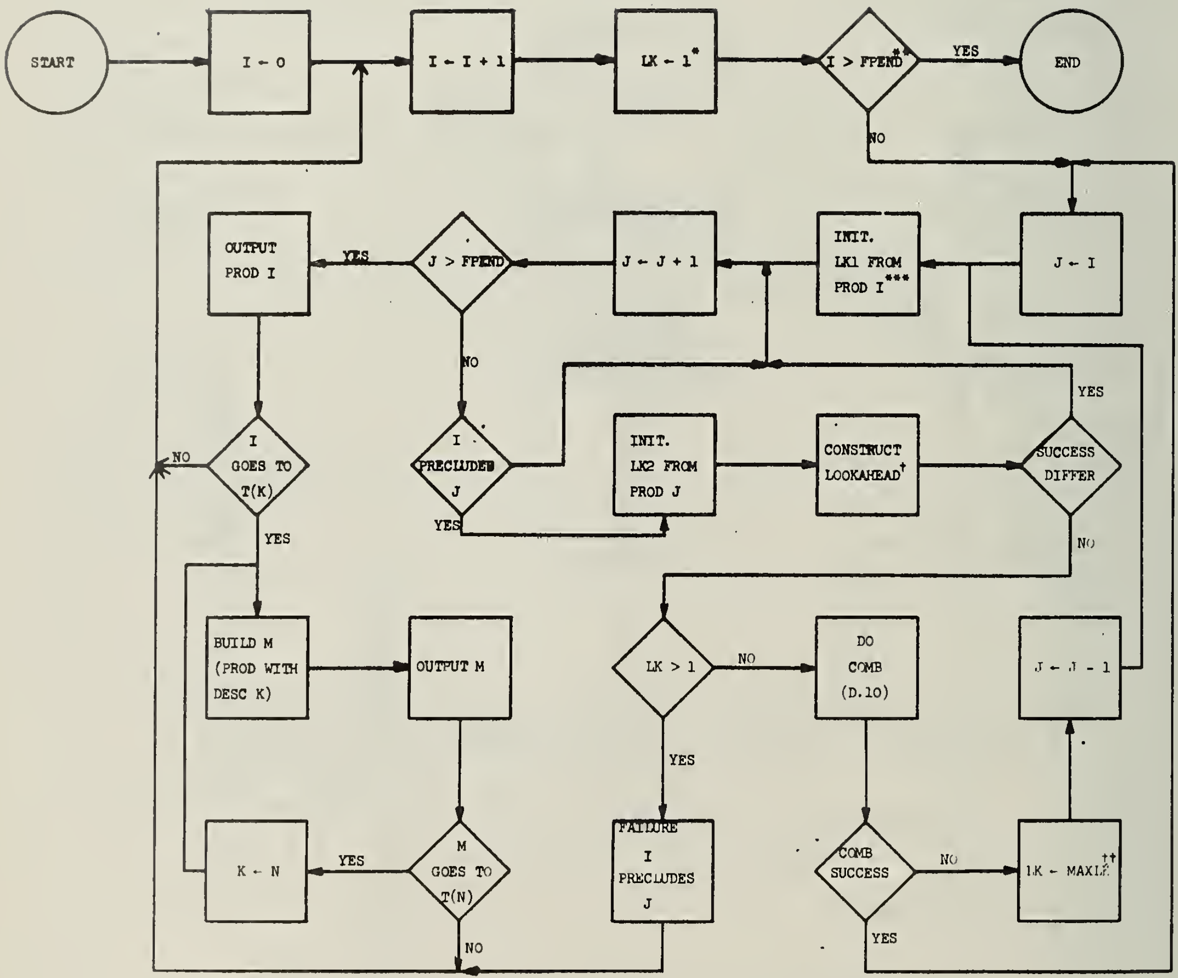
** same as (*) but using D_{NTH-X}

D.7 Construction of CTH-I and CNTH-I Groups



* CGP [I,J] is the J-th entry of combination I

D.8 Construction of CT-I Groups



* LK is the lookahead length to be used

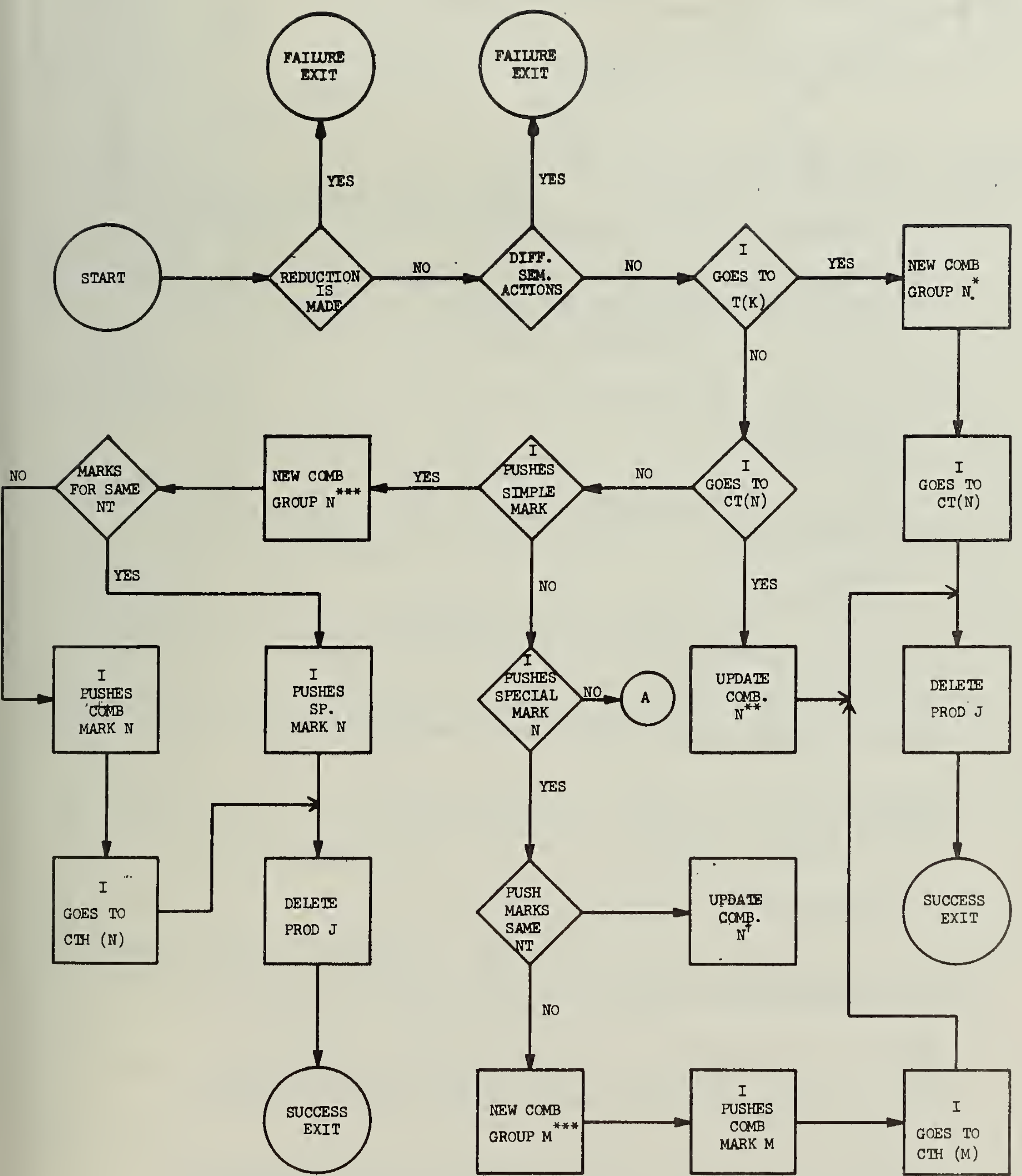
** FPEND is the last FPL production in a group

*** see Chapter Five

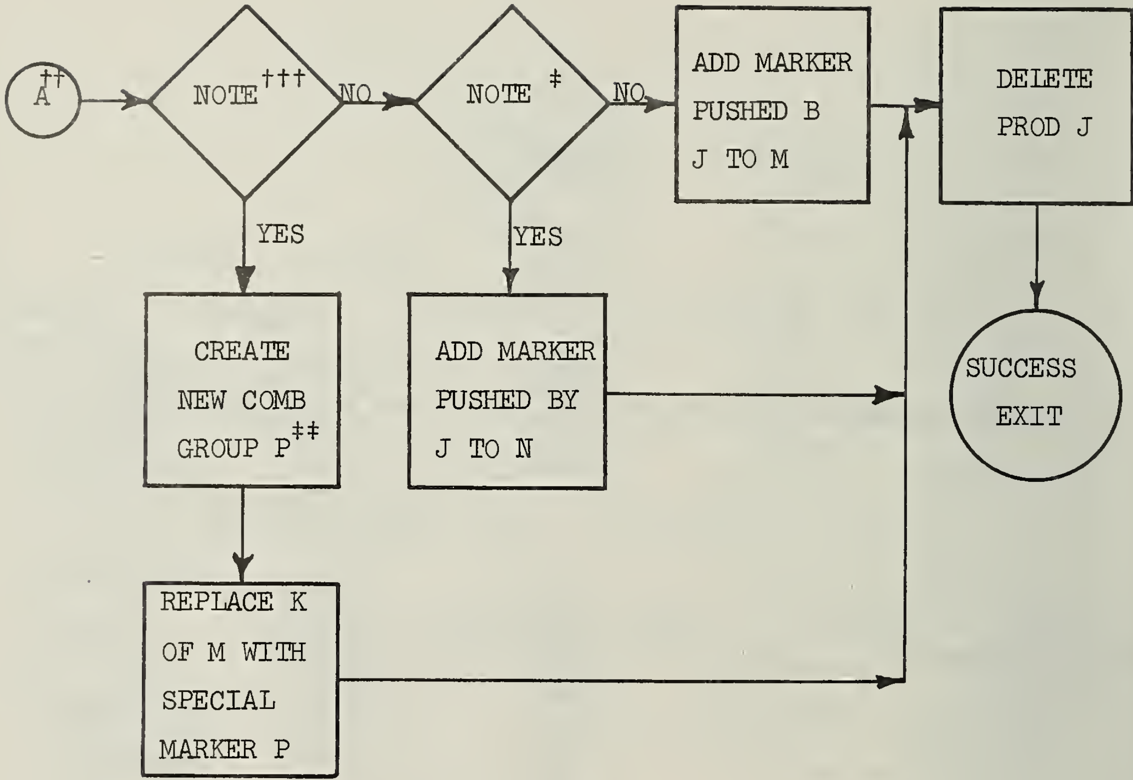
† see Figure 2 of Chapter Five

†† MAXLK is the length of the multisymbol lookahead

D.9 Preclusion Elimination and T (k) Production Generation



D.10 Combination of Productions (Part I)



* The transfer labels $T(K_I)$ and $T(K_J)$ from productions I and J are the entries

** add the transfer label $T(K_J)$ from production J

*** markers pushed by I and J are the entries

† add the marker pushed by J

†† I must push $(\bar{M})$

††† marker pushed by J is for the same nonterminal as the simple marker which is K-th entry of combination M

‡ same as ††† except K-th entry is special marker N

‡‡ K-th entry of M and bar pushed by J are the entries

D.10 Combination of Productions (Part II)

LIST OF REFERENCES

[1] Lewis, P. M. and Stearns, R. E., "Syntax-Directed Transduction", J. ACM 15, (July 1968), pp. 465-493.

[2] Feldman, J. A., "A Formal Semantics for Computer Oriented Languages", Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1964, Summarized in Comm. ACM 9, (Jan. 1966), pp. 3-9.

[3] Lucas, P. and Walk, K., "On the Formal Description of PL/1", ACM Symposium on Programming Languages Definition, San Francisco, California, Aug. 1969.

[4] McKeeman, W. M., et al, "The XPL Compiler Generating System", AFIPS 1968 Fall Joint Computer Conference, San Francisco, California, Aug. 1969.

[5] Wirth, N. and Weber, H. "EULER-A Generalization of ALGØL and its Formal Definition", Part I, Part II, Comm. ACM 9, (Jan., Feb. 1966), pp. 13-25, 89-99.

[6] Ingerman, P. Z., "A Syntax Oriented Translator", Academic Press, Inc., New York, 1966.

[7] Brooker, R. A, and Morris, D., "A General Translation Program for Phrase Structure Languages", J. ACM, 9 (Jan. 1962), pp. 1-10.

[8] Trout, R. G., "A Compiler-Compiler System", Proc. ACM 22nd National Conference, 1967, pp. 317-322.

[9] Floyd, R. W., "A Descriptive Language for Symbol Manipulation", J. ACM, 8 (Oct. 1961), pp. 579-584.

[10] Evans, A., "An ALGØL 60 Compiler", Annual Review in Automatic Programming, 4 (1964), pp. 87-124.

[11] Naur, P., et al, "Revised Report on the Algorithmic Language ALGØL 60", Comm. ACM, 6 (Jan. 1963), pp. 1-17.

[12] Earley, J. C., "Generating a Recognizer for a BNF Grammar", Computation Center Report, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1965.

[13] DeRemer, F. L., "On the Generation of Parsers for BNF Grammars: An Algorithm", ILLIAC IV Document No. 147, Department of Computer Science, University of Illinois, Urbana, Illinois, 1968.

[14] Knuth, D. E. , "On the Translation of Languages from Left to Right", Information and Control, 8 (Oct. 1965), pp. 607-639.

[15] DeRemer, F. L., Practical Translators for LR (k) Languages," Massachusetts Institute of Technology, Project MAC, Cambridge, Massachusetts, 1969.

[16] Korenjak, A. J., "A Practical Method for Constructing LR (k) Processors", Comm. ACM, 12 (Nov. 1969) pp. 613-623.

[17] Earley, J. C., "An Efficient Context-Free Parsing Algorithm", Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1968, Summarized in Comm. ACM, 13, (Feb. 1970), pp. 94-102.

[18] Hopcroft, J. E. and Ullman, J. D., Formal Languages and Their Relation to Automata, Addison-Wesley, Inc., Reading, Massachusetts, 1969.

[19] Floyd, R. W., "Bounded Context Syntactic Analysis", Comm. ACM, 7 (Feb. 1964), pp. 62-67.

[20] Mercer, R. L., "TWINKLE-A Syntax Language for a Translator Writing System", ILLIAC IV Document No. 218, Department of Computer Science, University of Illinois at Urbana-Champaign, 1970.

[21] Machado, N. E., "ISL - A Semantics Language for a Translator Writing System", ILLIAC IV Document No. 203, Department of Computer Science, University of Illinois at Urbana-Champaign, 1969.

[22] LaFrance, J. E., "Syntax-Directed Error Recovery for Syntax-Directed Compilers", Ph. D. Dissertation in Process, Department of Computer Science, University of Illinois at Urbana-Champaign.

[23] Horning, J. J. and Lalonde, W. R., "Empirical Comparison of LR (k) and Precedence Parsers", ACM, SIGPLAN Notices, 11, (Nov. 1970), pp. 10-24.

VITA

Alan James Beals was born on May 24, 1941 at Chicago, Illinois. He attended high school at Leyden Community High School in Franklin Park, Illinois, from which he was graduated in June, 1959. He attended undergraduate school for one year at Knox College in Galesburg, Illinois.

After two years in the United States Army, Mr. Beals completed his undergraduate education at the University of Illinois where he graduated in February, 1967, with a B.S. in Mathematics. He then began his graduate work at the University of Illinois where he received his Master's Degree in February, 1969. During the time spent earning his Master's Degree, and while working on his Ph. D., he was also a Research Programmer on the ILLIAC IV Project in the Department of Computer Science of the University of Illinois.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

IGINATING ACTIVITY (Corporate author) enter for Advanced Computation iversity of Illinois at Urbana-Champaign rbana, Illinois 61801	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP

PORT TITLE

IE AUTOMATIC GENERATION OF DETERMINISTIC PARSING ALGORITHMS

SCRIPTIVE NOTES (Type of report and inclusive dates)

esearch Report

THOR(S) (First name, middle initial, last name)

.an James Beals

PORT DATE .y 25, 1971	7a. TOTAL NO. OF PAGES 160	7b. NO. OF REFS 23
ONTRACT OR GRANT NO. SAF 30(602)-4144 ROJECT NO. RPA Order 788	9a. ORIGINATOR'S REPORT NUMBER(S) ILLIAC IV Document No. 248 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) DCS Report No. 458	

ISTRIBUTION STATEMENT

opies may be requested from the address given in (1) above.

UPPLEMENTARY NOTES one	12. SPONSORING MILITARY ACTIVITY Rome Air Development Center Griffiss Air Force Base Rome, New York 13440

BSTRACT

This paper describes an algorithm for thecconversion of a grammar n the form of a set of context free productions into a deterministic parsing lgorithm described by a set of modified Floyd productions. The algorithm s extended in such a way that it may easily become a part of a complete ranslator writing system and make use of the information available in the emantic part of such a system. The paper also includes a determination of ne subclass of context free grammars which can be successfully converted and comparison of this algorithm with some other methods of generating parsing lgorithms from context free grammars.

14. KEY WORDS	LINK A		LINK B		LIN[K C]
	ROLE	WT	ROLE	WT	ROLE
Compiler-Compilers					